UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Simulación con OMNeT++ de escenarios de reparto de canales en WIFI

Autor: Víctor Iglesias Alcón

Tutor/es: Iván Marsá Maestre

TRIBUNAL:

Presidente: José Manuel Giménez Guzmán

Vocal 1º: Enrique de la Hoz de la Hoz

Vocal 2º: Iván Marsá Maestre

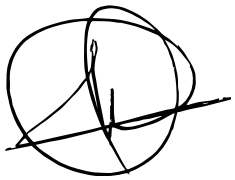
FECHA: 20 de Septiembre de 2016

Agradecimientos

Quiero dedicar esta memoria a todos aquellos que han estado a mi lado, durante todo este proceso. Para empezar quiero nombrar, de forma global, a todos los familiares, profesores y compañeros con los que he podido compartir, muchos buenos y malos momentos, durante toda la carrera. De todas estas personas, quiero destacar a José Manuel Giménez, quien me ha ayudado, en todo lo que estaba a su alcance, a lo largo de todo el trabajo de fin de grado y a mi tutor de este, Iván Marsá Maestre, con el que he podido contar siempre que lo he necesitado.

Quiero aprovechar esta oportunidad, para agradecer a todos los profesores con los que he tenido mayor relación, ya sea porque hemos tenido más horas de clase o porque han tenido que aguantarme en más de una tutoría. De estos tengo que destacar, a los anteriormente nombrados y sumar los siguientes: Enrique de la Hoz De la Hoz, Juan Antonio Carral Pelayo, Isaias Martínez Yelmo, Francisco Javier Ceballos Sierra, David Orden Martín, Elena Saiz Villanueva, Sonia Pérez Díaz, Andrés Navarro Guillén. Todos ellos y muchos más, que me dejo sin nombrar, han colaborado de forma indirecta con este trabajo, pero totalmente directa con mi formación a lo largo de estos años. He tenido la suerte de poder aprender muchos conocimientos de vosotros, por esta razón, quiero deciros que estoy muy satisfecho con vuestra labor. Gracias a todos, por enseñarme todo lo que sabéis, seguir así.

Como estudiante, creo que tengo un punto de vista, algo diferente. Esto se debe a que accedí a la universidad por medio de la prueba de acceso a mayores de 25 años, a la edad de 27. Esto me dio la oportunidad de vivir toda esta experiencia de una forma más madura y objetiva, que la mayoría de mis compañeros. Fui totalmente consciente del esfuerzo, que tienen que hacer muchos padres, para que, los hijos, podamos estudiar. Pues es a ellos a los que quiero dedicar este último párrafo, porque sin ellos nada de esto hubiera sido posible. Con mucho esfuerzo me han brindado la oportunidad de volver a estudiar, algo que tiene un valor incalculable. Muy poca gente tiene tal oportunidad y gracias a su esfuerzo, han hecho posible que hoy esté escribiendo esta memoria, con el fin de acabar mis estudios de Grado en Ingeniería Telemática. Gracias por todo. Espero, algún día, tener la oportunidad de hacer algo por vosotros, como lo que habéis hecho por mi.



Víctor Iglesias Alcón

Índice de contenidos

Índice de tablas.

Índice de capturas.

Índice de fórmulas.

1. INTRODUCCIÓN

1.1. Palabras clave.

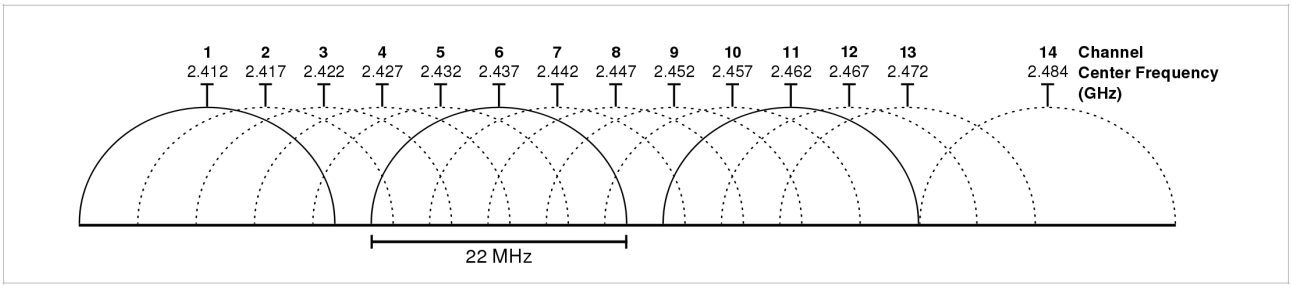
IEEE 802.11, espectro radioeléctrico, interferencias, frecuencia, coloreado, simulación, OMNeT++.

1.2. Resumen en castellano.

Cuando intentamos acceder a una red ***Wifi IEEE 802.11n***, en la mayoría de los casos lo hacemos a través de puntos de acceso, estos tienen asignado unos 20MHz de la banda total de 2,4GHz, a lo que se denomina canal. Habitualmente estos se asignan de manera aleatoria, lo que puede dar lugar a que algunos de estos puntos de acceso trabajen en el mismo canal y se interfieran mutuamente o incluso canales adyacentes que producen interferencias por solapamiento frecuencial. Para resolver este problema se proponen una serie de mecanismos para asignarlos de una forma inteligente. Este trabajo consiste en interpretar los parámetros de la simulación a generar en un proyecto para ***OMNeT++***. En este software se simula un escenario virtual donde se toman medidas sobre como se comportan los radio-enlaces o nodos que están interfiriendo unos con otros.

1.3. Abstract.

When we try to access a ***Wifi IEEE 802.11n*** network, we normally do it through what are called access points. These access points are assigned 20 MHz of the total 2.4GHz of available bandwidth. These 20 MHz "strips" of the total available 2.4 GHz of bandwidth are called channels. The problem is that the access points are usually assigned channels randomly. This can result in two different access points being assigned the same channel at the same time. This results in the two access points competing with each other for the channel. And this often results in the two access points interfering with each other and/or having overlapping signals. To fix this problem, we need to use an intelligent assignment system to assign each access point it's own channel. The work is to interpret the simulation parameters to generate a ***OMNeT++*** project. In this software a virtual scene is simulated where you are get measures such as radio links or nodes that are interfering with each other behave.



Captura 1: Representación de las bandas de frecuencias por canal (extraída de wikipedia).

1.4. Resumen extendido.

El objetivo a tratar de este trabajo es desarrollar una aplicación para poder simular un modelo analítico de red ***IEEE 802.11n (Wifi)*** en un simulador de eventos discretos desarrollado especialmente para trabajar con redes de comunicaciones, ya sean cableadas o inalámbricas, estas últimas son las más interesantes de cara a este trabajo. Este software recibe el nombre de ***OMNeT++***, los dos últimos "*+*" hacen referencia a que todo el programa y módulos del mismo se han desarrollado en ***C++***.

Para ponernos en situación a la hora de desarrollar el trabajo en cuestión, debemos entender cómo funciona el protocolo ***IEEE 802.11n***, comúnmente conocido como ***Wifi*** y centrarnos con mayor prioridad en los mecanismo destinados a la asignación de canales.

El problema que encontramos a la hora de asignar los canales en escenarios con varios puntos de acceso, es que si dos de ellos están dentro del mismo radio de cobertura y tienen asignado el mismo canal, generarán interferencias el uno sobre el otro, incluso si los canales son adyacentes pueden llegar a interferir también por solapamiento frecuencial. Para solucionar este problema se plantea generar una serie de mecanismos que asignen estos canales de una forma inteligente para poder alcanzar la mayor eficiencia del espacio radioeléctrico, que por necesidad tienen que compartir todos los dispositivos concentrados en un mismo área dada.

Estos mecanismos se han modelado basándose en un problema matemático conocido como *Coloreado de Grafos*, donde un conjunto de grafos representaría un red ***Wifi*** con distintos puntos de acceso con clientes conectados, cada uno de ellos recibe un color, el cual representaría un canal (frecuencia). Estos datos se almacenan en un archivo que debemos interpretar para poder transcribirlos en una escena de un simulador de eventos discretos que recibe el nombre de ***OMNeT++***.

OMNeT++ es el software que ofrece mayores facilidades para configurar un escenario con simples comandos. Por esta razón y por muchas otras fue el elegido para el desarrollo de este trabajo. El lenguaje en el que se ha desarrollado toda la aplicación es ***Python 2.7***. Las razones por la que se determinó que esta tecnología era la adecuada, es porque el modelo analítico está desarrollado en este mismo lenguaje y también este ofrece una gran versatilidad de módulos que resultan muy útiles a la hora de trabajar.

Para el desarrollo del proyecto fue necesaria una gran labor de investigación previa, ya que ***OMNeT++*** es un programa un tanto peculiar y no existe demasiada información en internet, por lo que requirió leer mucha documentación para poder empezar a realizar alguna simulación con éxito y así desarrollar la aplicación en cuestión. También fue necesario consultar algunos de los estándar definidos por ***IEEE*** con el fin de que el escenario a generar sea lo más fiable y cumpla con ellos.

Una vez finalizado el desarrollo queda una de las partes más importantes, hacer análisis de resultados y cerciorarnos de que todo se genera de forma correcta. Para esto se han generado varias simulaciones sobre el mismo escenario, concretamente 10 simulaciones por cada mecanismo de asignación simulado. En el apartado 4 de la sección 4, de esta memoria, se expondrán estos resultados obtenidos.

2. ASIGNACIÓN DE CANALES EN IEEE 802.11

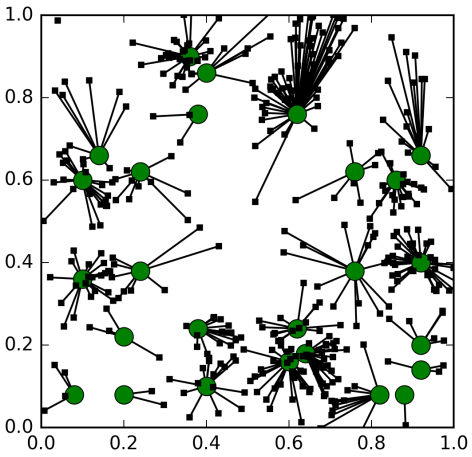
El problema consiste, básicamente, en minimizar las interferencias que se producen cuando distintos puntos de acceso cercanos están emitiendo en el mismo canal o adyacentes, es decir en la misma banda de frecuencias. Esto se conoce como solapamiento frecuencial. Pues bien, es aquí donde entra en juego todo el estudio de técnicas de negociación automática para la asignación de canales (frecuencias). Para solventar este problema se propone una serie de mecanismos de negociación no lineales que veremos con más detalle a continuación.

2.1. Modelado del problema.

Como se ha mencionado anteriormente, esto ha sido modelado basándose en un problema matemático conocido con el nombre de *Coloreado de Grafos*, ya que guarda cierta relación entre color y frecuencia. Pertenece al campo de estudio *Teoría de Grafos*, que de una forma muy básica se podría definir como un conjunto de vértices unidos con aristas entre ellos.

2.1.1. Modelado abstracto basado en coloreado de grafos.

El problema clásico de coloreado de grafos, trata de un caso especial de grafo. En el que existen etiquetas que reciben el nombre de color donde los vértices y aristas adyacentes, se colorean de manera que no reciban el mismo color. En el problema que nos ocupa, las frecuencias juegan el papel de colores.



Captura 2: Grafo de asociación de la Escuela Politécnica Superior.

Para este trabajo se propone un grafo que modela con más fidelidad el problema. En este grafo podemos distinguir dos tipos de vértices o nodos: los puntos de acceso (AP) y los clientes (C). En el caso de las aristas también se pueden distinguir dos tipos: aquellas que representan la asociación entre puntos de acceso con sus clientes asociados (AP-C) y aquellas que simulan las interferencias entre nodos, estas unirían a aquellos que se encuentren dentro del radio de cobertura (R), pudiendo distinguir tres casos: entre puntos de acceso (AP-AP), entre punto de acceso y cliente (AP-C), que sólo se conectan si no están asociados entre sí y entre clientes (C-C), que se unirán cuando no estén asociados al mismo punto de acceso.

Para incluir en este modelo la potencia de la interferencia entre nodos, se asigna un peso a cada arista, este valor se basa en tres factores.

Lo primero se asigna un peso a cada par de colores que representan las interferencias entre colores o canales. En esta extensión del problema se consideran estas entre colores adyacentes que se encuentran a cierta distancia.

Lo segundo se modelan estos pesos, pero en este caso se tiene en cuenta la distancia entre nodos involucrados, así las aristas obtendrán diferentes pesos en función de la cercanía de estos nodos. Esto hace que nuestro grafo no sea abstracto si no geométrico puesto que los vértices están en una posición determinada.

Por último se tienen en cuenta los intervalos de tiempo activos, puesto que los nodos no se encuentran activos en todos los instantes, si no que alternan entre estados y los que se encuentran inactivos no interfieren en el coloreado.

En la captura número 2 aparece un grafo directamente dibujado con herramientas que proporciona ***NetworkX***. Este representa la planta baja de la *Escuela Politécnica Superior* y ha sido objeto de estudio para el desarrollo del trabajo.

2.1.2. Efectos de la propagación en la cobertura.

Para que los datos sean lo más fieles al modelo de ***IEEE 802.11n***, debemos tener en cuenta tanto el radio de cobertura como el de interferencias. Para esta labor se definen las pérdidas de potencia expresadas en dB y las antenas a una altura que va de 1 a 2,5 metros.

$$P_{loss} = 40 \cdot log_{10}(d) + 20 \cdot log_{10}(f) - 20\ log_{10}(ht \cdot hr)$$

Fórmula 1: Pérdidas de potencia en dB.

De esta ecuación podemos despejar el radio de interferencias, el cual es muy importante, puesto que es el que define los vértices que estarán conectados en el grafo de interferencias.

$$R = 10^{\frac{Pt + Gt + Gr - S - 7{,}6 + 20 \cdot log_{10}(ht \cdot hr)}{40}}$$

Fórmula 2: Radio de cobertura en metros.

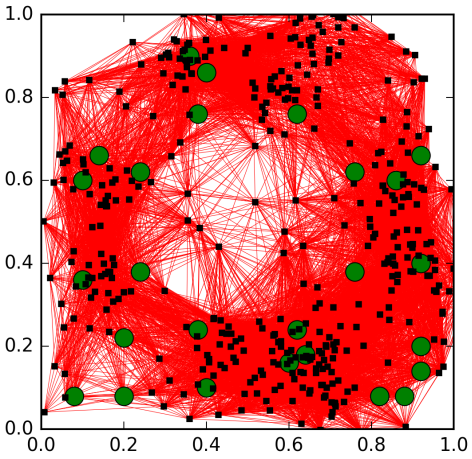
Este caso se expondrá con más detalle en el apartado 4.1.1., este aborda todo lo relativo a la investigación previa y el radio de interferencias fue necesario investigar alguna dato, para concretar que todo estaba correctamente interpretado en el escenario a generar.

2.1.3. Efectos de las interferencias.

En el modelo, a la hora de medir la calidad de la señal recibida, hay que tener en cuenta todas las señales que interfieren, por lo que resulta necesario medir la potencia de estas, tanto en puntos de acceso como en clientes. En este modelo sólo se tienen en cuenta las

interferencias que generan otros nodos ***IEEE 802.11***, sin tener en cuenta posibles interferencias por otros dispositivos que emiten en la misma frecuencia de 2,4GHz, ya que se considera que no forman parte del problema, también se han despreciado todas las interferencias que provengan de nodos que estén fuera del radio de cobertura. Por otro lado se ha asumido que los clientes asociados al mismo punto de acceso no interfieren entre sí, suponiendo que es el propio punto de acceso el que gestiona esto.

La captura 3 muestra las aristas que representan las interferencias entre nodos, creando así el grafo de estas, el cual también ha sido objeto de estudio para el desarrollo de este trabajo.

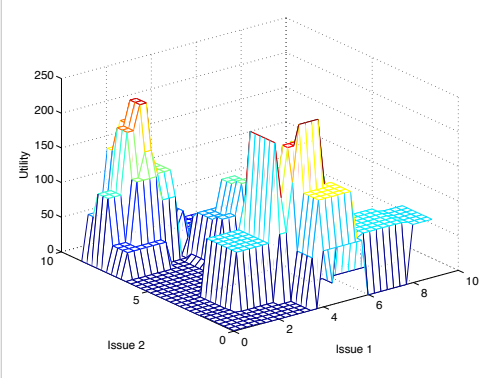


Captura 3: Grafo de interferencias de la Escuela Politécnica Superior.

2.2. Técnicas de negociación automática.

Existen varias técnicas de negociación automática, pero aquí se plantea un problema típico de negociación, que consta de tres partes principales. El dominio de la negociación, el protocolo de interacción y los mecanismos de decisión. Estas se expondrán con mayor detalle a continuación.

2.2.1. Dominio de la negociación.



Captura 4: Función de utilidad.

En este trabajo se ha asumido un dominio de negociación multiatributo, para el cual hay un acuerdo o solución que contiene una serie de atributos (*issues*). Para *n* puntos de acceso *nAP* y un acuerdo *S*, el dominio viene dado por la siguiente expresión $S = \{ S_i \mid i \in 1,\ldots,nAP \}$ donde cada S_i es el numero de canal ***IEEE 802.11n*** asignado al punto de acceso *i*. Para este trabajo se asume la banda de 2,4GHz con un máximo de 11 canales de 20MHz, en vez de los 40MHz que soporta el protocolo ***IEEE 802.11n***, que puede albergar hasta 13 canales, pero estos no se usan en todos los países.

Para este estudio los puntos de acceso se dividen en dos proveedores P_1 y P_2. Cada uno de ellos se encarga de la asignación de frecuencias en sus propios puntos de acceso, de este modo P_1 y P_2 son los encargados de actuar como agentes de la negociación de frecuencias. Por último estos agentes usarán una función de utilidad para una solución o acuerdo según el modelo expuesto anteriormente y solamente teniendo en cuenta aquellas interferencias de los puntos de acceso que le corresponden. Según la hipótesis de los involucrados en este trabajo, todo esto hará que las funciones de utilidad sean altamente complejas, con muchos máximos y mínimos locales. La captura 4 muestra un ejemplo de esta función.

2.2.2. Protocolo de interacción.

Existen muchos protocolos de interacción, pero para el caso que nos ocupa se ha empleado un protocolo de mediación simple (simple text mediation) que consta de los siguientes cuatro pasos:

1) Se genera un contrato al azar S_0^c, que sería el primer candidato, en este caso el contrato es el número de canal.
2) El agente que actúa como mediador propone un contrato S_t^c cada iteración *t*.
3) Todos los agentes votan a favor o en contra del contrato propuesto por el mediador.
4) A partir de los contratos anteriores y votos recibidos el mediador genera un nuevo contrato S_{i+1} y se reanuda desde el segundo paso.

El proceso se repite hasta alcanzar el máximo de iteraciones o al llegar a una situación de parada. Con esto tendríamos ya un contrato entre los agentes implicados, pero esto debe completarse con mecanismos de decisión o estrategias de los agentes que actúan como negociadores y las de aquellos que actúan como mediador.

2.2.3. Mecanismos de decisión.

El mediador es el encargado de ejecutar el mecanismo de decisión, este consiste en generar contratos candidatos de la siguiente manera:

- Si ambos agentes han votado a favor del contrato S_t^c generado, este se emplea como contrato base S_b para generar el siguiente contrato S_{i+1}.
- Para generar S_{i+1} se escoge un punto de acceso al azar y un canal aleatorio para este.
- Al final de realizar un número determinado de iteraciones el mediador anuncia el que sería el contrato final que se trata del último aceptado por ambos agentes.

Para este estudio se han empleado dos mecanismos diferentes para generar los votos de los dos agentes sobre cada contrato candidato S_c:

- **Hill-climber (HC):** Este agente únicamente votará a favor en el caso de que el contrato en cuestión sea mejor que el anterior que hayan aceptado ambos agentes. En el inicio, cuando no existe un contrato previo aceptado, siempre se votará a favor.

- **Simulated Annealing (SA):** Este agente emplea una técnica no lineal conocida como Simulated Annealing. En el caso de que el contrato sea peor que el anterior aceptado por ambos agentes, existe una posibilidad de que sea aceptado por estos. Esta probabilidad P_a dependerá de dos parámetros, la pérdida de utilidad *Δu* y la temperatura de *annealing T*, tal que $P_a = e^{-\Delta u/T}$. La temperatura está fijada a un valor determinado en el inicio y decrece de manera lineal por cada iteración.

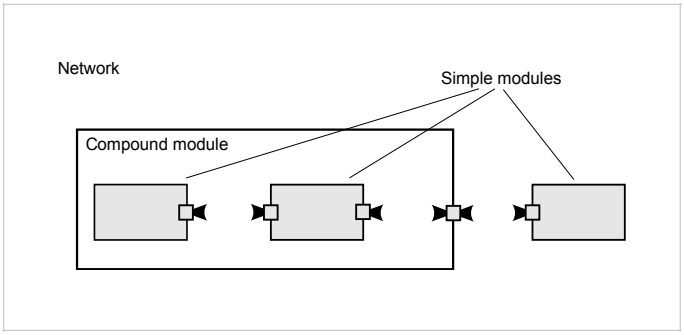
3. SIMULADOR OMNET++.

Este software se trata de un simulador de eventos discretos diseñado en la *Universidad Técnica de Budapest*. Su enfoque principal está orientado a redes de comunicación aunque también tiene otras aplicaciones. Se basa en componentes modulares o módulos que a su vez pueden estar formados por otros módulos denominados submódulos. Estos módulos están escritos en lenguaje ***C++*** y se compilan de forma independiente. Estas características hacen que ***OMNeT++*** esté ganando mucha popularidad.

Es un software de licencia pública académica, por lo que no puede usarse con fines comerciales. También hay que destacar que se trata de un software open source y su código fuente puede ser consultado o modificado si fuera necesario.

3.1. Módulos.

Un modelo de simulación ***OMNeT++*** consiste en módulos que se comunican por medio de mensajes. Los módulos se pueden dividir en dos grandes variantes, los módulos simples y por otro lado los compuestos. Los módulos compuestos están formados por otros módulos a los que nos referiremos como submódulos, todos los módulos de tipo red (network) son en sí mismo módulos compuestos. Todos estos están escritos en ***C++*** utilizando la biblioteca de clases de simulación. Los mensajes pueden ser enviados a través de conexiones que abarcan módulos o directamente a otros módulos.



Captura 5: Módulos simples y compuestos.

Un modelo para simular en ***OMNeT++*** consiste en módulos jerárquicamente anidados que se comunican entre sí por medio de envío de mensajes a través de puertas de entrada-salida *"I/O"* conectadas entre sí.

Un módulo puede contar con cuatro estructuras de datos principales: los parámetros (parameters), las puertas (gates), las conexiones (connections) y por último los submódulos (submodules).

Las puertas (gates, en inglés) son interfaces de entrada-salida para el envío de mensajes de un módulo a otro o incluso pueden ser enviado a sí mismo, lo que se denomina un auto-mensaje. Estas puertas se conectan entre sí a través de conexiones que pueden aplicar varios protocolos y velocidades de transmisión.

La estructura de un modelo de simulación se representa como una red, se escribe en un lenguaje propietario de ***OMNeT++*** denominado ***NED*** (*Network Description*).

3.2. Lenguaje NED.

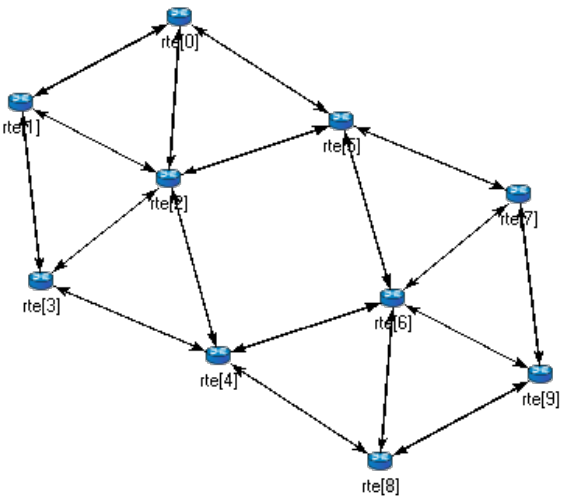
El lenguaje ***NED*** está destinado a la interfaz de módulos para ***OMNeT++***, con él se describen todos los elementos que componen la red a simular. El lenguaje ***NED*** consta de una jerarquía en árbol que puede ser serializado en *XML*.

NED permite declarar módulos simples, conectar y agruparlos en módulos compuestos. Los canales son otro tipo de componente, cuyas instancias también pueden ser utilizadas en módulos compuestos.

La forma tradicional de hacer frente a la complejidad es mediante la introducción de jerarquías. En ***OMNeT++***, cualquier módulo que fuese demasiado complejo como una entidad única, puede dividirse en módulos más pequeños y se utiliza como un módulo compuesto.

```
//
// A network
//

network Network{
    submodules:
        node1: Node;
        node2: Node;
        node3: Node;
        ...
    connections:
        node1.port++ <--> {datarate=100Mbps;} <--> node2.port++;
        node2.port++ <--> {datarate=100Mbps;} <--> node4.port++;
        node4.port++ <--> {datarate=100Mbps;} <--> node6.port++;
        ...
}
```

Captura 6: Código NED que genera la red.



Captura 7: Red descrita por el lenguaje NED.

3.3. Archivos de inicialización.

Para poder simular la red en cuestión, es necesario generar un archivo de inicialización donde como su nombre indica se inician los parámetros de los módulos que forman la red. Este archivo puede llevar varias secciones, pero hay una que es obligatoria, esta es la sección general. En cada una se ajustan los parámetros para cada escenario, pero todas las secciones extras heredan los ajustes de la sección general.

```
[General]
network = Graph80211
...
```

Captura 8: Ajustando la red a simular en la sección general del archivo omnetpp.ini.

```
[Config RoutedWlansUDP]
description = "Simulando trafico UDP"
**.packetSize = 120B
**.printPing = true
...
```

Captura 9: Sección personalizada para simular tráfico UDP en el archivo omnetpp.ini.

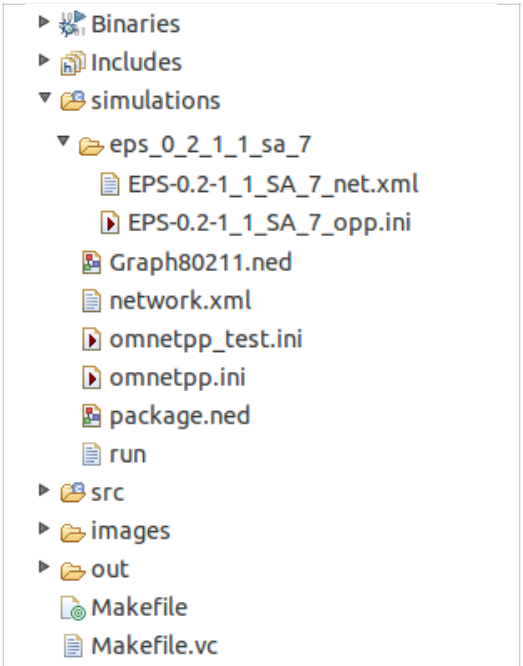
```
**.host[0].numUdpApps = 1
**.host[0].udpApp[*].typename = "UDPVideoStreamSvr"
**.host[0].udpApp[*].sendInterval = 10ms
**.host[0].udpApp[*].localPort = 3000
...
```

Captura 10: Ajustando parámetros de la simulación en el archivo omnetpp.ini.

3.4. Proyecto de OMNeT++.

Un proyecto de ***OMNeT++*** debe constar como mínimo de un archivo de inicialización y un archivo de descripción de red ***NED***, todo lo demás es el propio ***OMNeT++*** quien se encarga de añadirlo o generarlo. Normalmente todo el proyecto se agrupa en varios directorios, "*Simulations*" donde podemos encontrar los archivos *.ini*, así como *XML* para varias configuraciones y los destinados a la interfaz *.ned*. Otro directorio nombrado como "*src*" esta destinado a guardar todos los archivos fuente escritos en ***C++*** para ser compilados. Además de estos puede contener más directorios para imágenes, plantillas, etc …

Dentro de un mismo proyecto podemos albergar varias simulaciones pudiendo agrupar estas dentro de subdirectorios.



Captura 11: Proyecto de OMNeT++.

4. OMNET++ Graph Simulation (OGSimulation).

El trabajo se puede dividir en cuatro etapas: investigación, planificación, desarrollo y por último un análisis de resultados. En esta sección se expondrá todo este proceso etapa por etapa. Todo el proceso ha requerido una duración de seis a siete meses a una dedicación media de 18 horas por semana. A continuación el diagrama de Gantt representa, de manera aproximada, la duración de cada una de estas etapas por semanas.

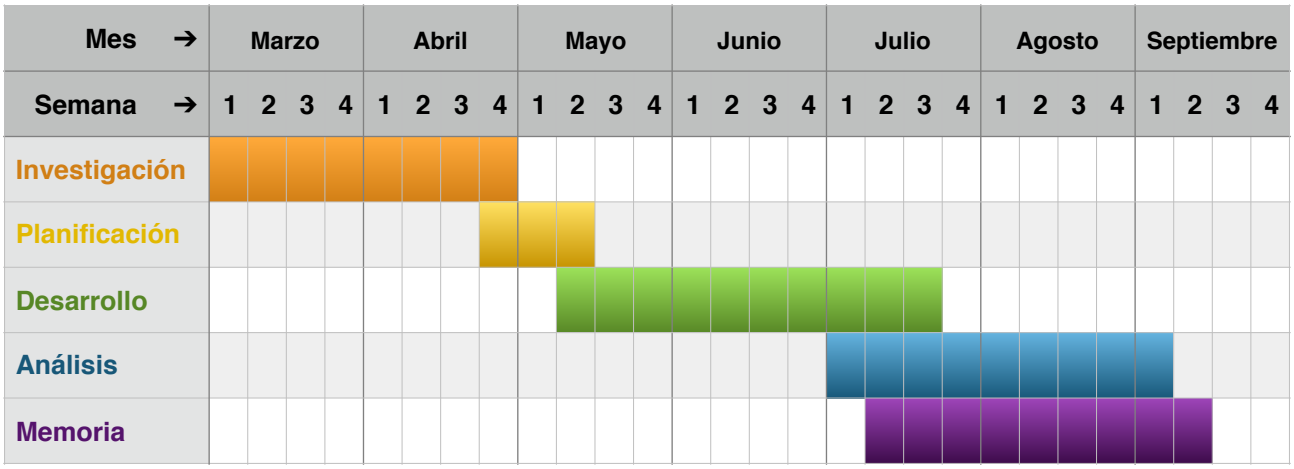


Tabla 1: Diagrama de Gantt del tiempo real empleado.

4.1. Investigación previa.

Esta labor fue una de las etapas del proyecto en la que más tiempo fue necesario emplear, pero es el primer paso para todo lo demás y completamente necesario para poder analizar todas las herramientas que tendremos a nuestra disposición, con el fin de conseguir nuestro objetivo de la manera más precisa posible.

A continuación expondremos de forma detallada toda la información que fue necesario consultar por cada tecnología involucrada en el trabajo.

4.1.1. IEEE 802.11n.

En la sección número dos se explican estos conceptos, pero ahora se expondrán aquellos que realmente fue necesario documentarse para entender el problema y así poder empezar a trabajar en el desarrollo.

En lo referente a este proyecto debemos prestar mayor atención a la asignación de canales, esto implica analizar los radios de propagación e interferencias de una antena estándar ***Wifi***. Para analizar estos problemas se facilitaron una serie de ecuaciones para poder comparar con los datos de las simulaciones a realizar en ***OMNeT++.*** La fórmula número 1, representa el cálculo de pérdidas para una antena transmisora y otra receptora que se encuentren próximas al suelo, concretamente a una altura comprendida entre 1 a 2,5 metros.

La potencia perdida ***Ploss***, expresada en dB, depende de la distancia entre antenas ***d***, expresada en metros, la frecuencia de la portadora ***f***, expresada en GHz y a las alturas de las antenas, transmisora ***ht*** y receptora ***hr***, expresadas ambas en metros, todos estos datos pueden ser ajustados en ***OMNeT++*** a excepción de la altura de las antenas, las cuales no se indica en ningún momento, pero el resultado a la hora de calcular el radio de cobertura, es idéntico si fijamos esta altura a 2 metros.

$$P_{loss} = 40 \cdot log_{10}(d) + 20 \cdot log_{10}(f) - 20\ log_{10}(ht \cdot hr)$$

Fórmula 1: Potencia de pérdidas en dB

Para analizar esto se estima en un caso límite, en aquel que la potencia recibida es exacta a la que corresponde con la sensibilidad del receptor, como se indica en la formula número 2, donde ***S*** es la sensibilidad del receptor, expresada en dBm, ***Pt*** es la potencia transmitida, expresada en dBm, ***L*** son las pérdidas debido a obstáculos, expresada en dB, ***Gt*** y ***Gr*** que representan la ganancia de antena, expresadas en dB, transmisora y receptora respectivamente.

$$P_t + G_t + G_r - L - P_{loss} = S$$

Fórmula 3: Sensibilidad del receptor en dB.

Si sustituimos la fórmula número 1 de ***Ploss*** en la fórmula número 4 podemos despejar el radio ***R*** como indica la fórmula número 2, si escogemos como sensibilidad del receptor un valor de -90 dBm, la altura de ambas antenas a 2 metros con ganancia de 10 dB para cada una de ellas y potencia transmitida de 30 mW (14,77 dBm), todo esto en un escenario sin obstáculos, como sucede en una simulación simple de ***OMNeT++***, el resultado que se obtiene es de 1721 metros, un valor que en un principio parece excesivo, pero curiosamente es muy similar al que nos ofrece ***OMNeT++*** ajustando estos mismos valores.

$$R = 10^{\frac{P_t + G_t + G_r - S - 7{,}6 + 20 \cdot log_{10}(ht \cdot hr)}{40}}$$

Fórmula 2: Radio de cobertura en metros.

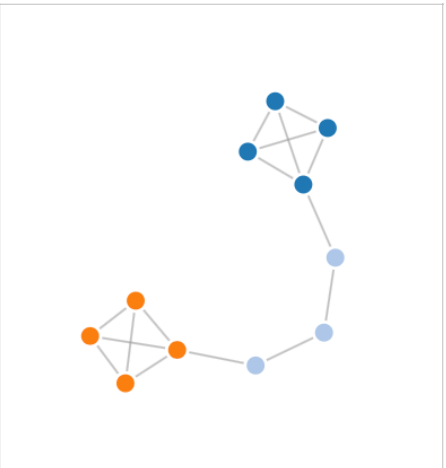
4.1.2. NetworkX.

Esta librería escrita en ***Python*** diseñada para el estudio de redes, se basa en el campo matemático de la teoría de grafos, por esta razón y porque el estudio, por parte del equipo que guía este trabajo de la *Universidad de Alcalá de Henares*, está desarrollado haciendo uso de esta misma, lo que hizo necesario investigar algunos datos de esta.

La teoría de grafos se puede describir de una manera muy básica como un conjunto de vértices unidos por aristas que forman uno o más grafos entre ellos.

Las redes pueden ser estudiadas como un grafo, esto hace que ***NetworkX*** sea una gran herramienta para el estudio de las mismas.

La teoría de grafos es un campo de estudio de las matemáticas muy amplio por lo que nos centraremos en la parte que nos interesa en este momento. Existen varios tipos de grafos y todos ellos pueden ser: dirigidos o no, simples, multigrafos, ponderados, etc …, pero para este estudio se centrará en los grafos unidireccionales.



Captura 12: Grafo generado con NetworkX

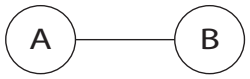
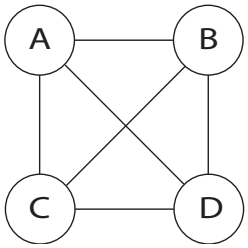
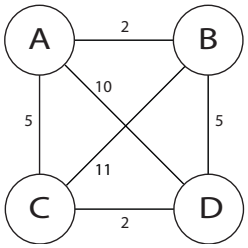
Simple	multigrafo	Simple dirigido	Ponderado
			

Tabla 2: Algunos ejemplos de tipos de grafos.

Un grafo se define como ***G = (V,E)***, que contiene ***V*** vértices y ***E*** aristas, el caso ponderado consiste en un grafo simple o compuesto, dirigido o no, pero el cual las aristas tienen un peso es decir cada camino tiene un coste o valor, lo que hace que estos sean muy útiles en el ámbito de las redes informáticas.

NetworkX nos ofrece una serie de clases para generar grafos con sus respectivos métodos para tratarlos como si de redes se tratasen. Esta librería nos brinda la facilidad de poder abstraernos de esta parte matemática y poder dedicarle más recursos a nuestra tarea principal, mientras ella se encarga de generar la información necesaria para poder representar el grafo a tratar en una escena a simular en ***OMNeT++***.

NetworkX es una herramienta muy extensa por lo que no se ha requerido investigar sobre de todas sus ventajas y desventajas, por esto se centrará la atención en la parte que sí fue necesario hacerlo para poder conseguir el objetivo en cuestión. Toda la información descrita a continuación ha sido extraída de la propia documentación ofrecida online del sitio oficial de esta herramienta networkx.github.io.

Para poder empezar a sacar partido a esta librería basta con descargarla de su sitio web, networkx.github.io, e importarla como si de cualquier otro módulo se tratase.

```
>>> import networkx as nx
```

Captura 13: Importando NetworkX.

En ***NetworkX*** existen algunos tipos de grafos, cada uno de ellos corresponde a una clase escrita en ***Python*** con sus respectivos métodos.

Tipos de grafos implementados en NetworkX	
nx.Graph()	Clase que genera un grafo unidireccional y no permite bucles entre el mismo nodo, es decir no pueden crearse conexiones entre el mismo nodo.
nx.DiGraph()	Clase que genera un grafo dirigido, grafos que tienen una determinada dirección, esta es una subclase de nx.Graph().
nx.MultiGraph()	Clase que genera un multigrafo unidireccional, grafo que contiene multiples conexiones entre pares de nodos, esto puede influir en el rendimiento sin ser al go muy significativo.
nx.MultiDiGraph()	Una versión de nx.MultiGraph(), pero que genera un multigrafo dirigido.

Tabla 3: Tipos de grafos en NetworkX.

Para crear una instancia de una de ellas sólo haría falta llamar al constructor de la clase y guardar la referencia en una variable.

```
>>> G=nx.Graph()
>>> G=nx.DiGraph()
>>> G=nx.MultiGraph()
>>> G=nx.MultiDiGraph()
```

Captura 14: Creando instancias vacías que representan los distintos tipos de grafos.

Lo primero es elegir qué tipo de grafo debemos usar. Para esto tenemos que saber cual es nuestro objetivo e informarnos de que tipo es el más conveniente. Para este caso se ha elegido el tipo unidireccional (nx.Graph), debido a que el material facilitado por la universidad está desarrollado utilizando este tipo de grafo.

A la hora de crear una instancia del grafo existen varias maneras de hacerlo:

- Generar el grafo mediante el algoritmo estándar para tipologías de red.
- Importando datos preestablecidos.
- Añadiendo vértices y aristas manualmente.

```
>>> import networkx as nx
>>> G=nx.Graph()
>>> G.add_edge(1,2)  # por defecto el valor de la arista = 1
>>> G.add_edge(2,3,weight=0.9) # valor específico
```

Captura 15: Tres formas de crear un grafo.

Los atributos de aristas y vértices pueden almacenar cualquier valor:

```
>>> import math
>>> G.add_edge('y','x',function=math.cos)
>>> G.add_node(math.cos) # Cualquier función puedes ser un vértice
```

Captura 16: Añadiendo aristas y vertices con funciones como valor.

Se pueden incluir múltiples aristas:

```
>>> elist=[('a','b',5.0),('b','c',3.0),('a','c',1.0),('c','d',7.3)]
>>> G.add_weighted_edges_from(elist)
```

Captura 17: Método para añadir un array de aristas.

NetworkX utiliza dict-of-dicts-of-dicts (diccionario de diccionarios de diccionarios) como base para la estructura de datos de la red, esto permite que las iteraciones en bucles se realicen rápidamente incluso en redes de grandes dimensiones.

Algunas ventajas de una estructura de datos dict-of-dicts-of-dicts:

- Encuentra y elimina vértices con dos búsquedas de diccionario.
- Mejor que “List” en operaciones de búsqueda con almacenamiento escaso.
- Mejor que “sets” ya que los los datos pueden ser adjuntados a las aristas.
- G[u][v] retorna el atributo diccionario de una arista.
- `n in G` comprueba si el vértice o nodo `n` esta contenido en el grafo G.
- `for n in G:` iteraciones a través del grafo G.
- `for nbr in G[n]:` iteraciones a través de los nodos más cercanos o vecinos.

Ejemplo de grafo unidireccional con las aristas (‘A’,’B’), (‘B’,’C’):

```
>>> G=nx.Graph()
>>> G.add_edge('A','B')
>>> G.add_edge('B','C')
>>> print(G.adj)
{'A': {'B': {}}, 'C': {'B': {}}, 'B': {'A': {}, 'C': {}}}
```

Captura 18: Representación de grafo unidireccional.

Los grafos contienen un atributo de tipo diccionario por cada arista. En el se guarda una estructura de datos dict-of-dicts-of-dicts y el diccionario interior almacena la relación entre nombre y valor.

```
>>> G=nx.Graph()
>>> G.add_edge(1,2,color='red',weight=0.84,size=300)
>>> print(G[1][2]['size'])
300
```

Captura 19: Creación y acceso de etiquetas en aristas.

4.1.3. INET OMNeT++.

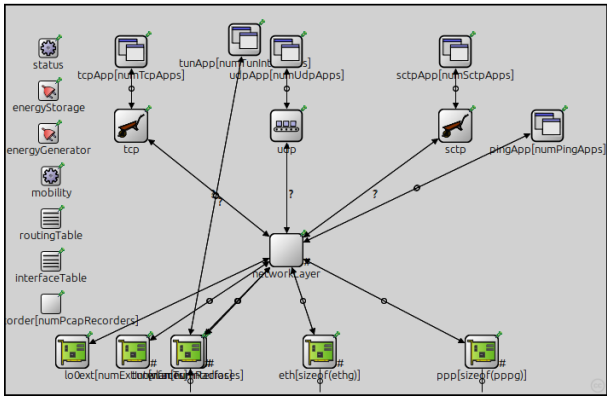
En la parte de ***OMNeT++*** que fue necesario documentarse previamente, ha sido expuesta en el apartado dedicado a él, por este motivo esta sección se centrará más en lo que ***INET*** se refiere. Sobre este framework la investigación se extendió algo más que la fase previa, puesto que hasta el último momento ha sido necesario consultar su referencia, la cual esta accesible online, es muy completa y está muy bien organizada.

INET es un framework para ***OMNeT++*** el cual añade un conjunto de módulos y submódulos que simulan los comportamientos de los estándares de red, más usados en la actualidad, en el ámbito de redes de comunicación. Para este trabajo lo que realmente nos hacía falta es entender todo aquello relacionado con el protocolo ***IEEE 802.11x***, una tarea que requiere tiempo ya que existe una gran cantidad de modelos relacionados con este dentro de ***INET***.

Después de muchos intentos fallidos a la hora de simular un escenario en ***OMNeT++*** se realizó con éxito una simulación utilizando los módulos, "***WirelessHost***" como cliente, "***AccessPoint***" como punto de acceso y "***Ieee80211DimensionalRadioMedium***" como medio, que se encarga de gestionar la capa física de la red a simular, en este caso es el espacio radioeléctrico. Estos tres módulos fueron escogidos para el trabajo y serán expuestos de forma más extensa a continuación.

WirelessHost

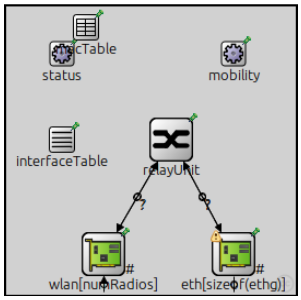
Este módulo representa un host estándar *(**StandardHost**)* que está equipado con dos interfaces, una ***Wifi*** *(**Ieee80211Nic**)* y otra *Ethernet*, esta última no será de gran interés para este trabajo. La interfaz ***Wifi*** dispone de un submódulo encargado de gestionar algunos de sus valores, tan importantes como la dirección *MAC* del punto de acceso asociado o el canal de este mismo.



Captura 20: Módulo WirelessHost.

AccessPoint

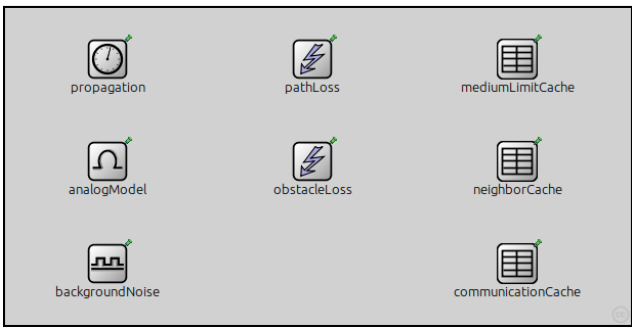
Este módulo representa un punto de acceso, también está equipado con estas dos interfaces y en este caso ambas están siendo utilizadas, ya que los clientes se conectan a él a través de la interfaz ***Wifi***, pero cada punto de acceso se conecta a su vez con el router principal mediante una interfaz *Ethernet*. Esta sólo ofrece conectividad entre puntos de acceso a través de enlace. Fue necesario incluir un router para poder conectar aplicaciones basadas en *IP* entre clientes que están asociados a distintos puntos de acceso.



Captura 21: Módulo AccesPoint.

Ieee80211DimensionalRadioMedium

Como se ha mencionado anteriormente, este módulo se encarga de gestionar la capa física del escenario que representará la red simulada, que en el caso del protocolo ***Wifi IEEE 802.11n*** se trata del espectro radioeléctrico. Es en este módulo en el que se ajustan los parámetros que caracterizan este medio. En el caso que nos ocupa se ha optado por dejar la mayoría de estos parámetros en el valor por defecto, los cuales se basan en los estándares definidos por ***IEEE 802.11n***.



Captura 22: Módulo Ieee80211DimensionalRadioMedium.

Durante todo el desarrollo ha sido necesario consultar en varias ocasiones toda la documentación que ***INET*** nos ofrece en su sitio web inet.omnetpp.org, con el fin de alcanzar nuestro objetivo de la forma más precisa.

4.2. Planificación.

En este apartado se explica con detalle la planificación necesaria previa a la fase de desarrollo. En ella se describe por qué se seleccionaron algunas herramientas en lugar de otras. El último punto de este apartado está dedicado a exponer un presupuesto estimado para el desarrollo.

4.2.1. Estimación del tiempo.

La estimación de tiempo siempre es una tarea compleja y difícil de concretar con exactitud, siendo muy común que el tiempo real empleado se dilate con respecto al que se estimó en un principio, esto se debe a que cuando se desarrolla una tarea tan especifica, en el camino siempre, o casi siempre, se suelen encontrar imprevistos. En la Tabla 4 representa este tiempo con un diagrama de Gantt, el cual difiere seis semanas con el real expuesto al inicio de esta sección, un tiempo nada despreciable.

Mes →	Marzo				Abril				Mayo				Junio				Julio			
Semana →	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Investigación																				
Planificación																				
Desarrollo																				
Análisis																				
Memoria																				

Tabla 4: Diagrama de Gantt del tiempo estimado en la etapa de planificación.

Se puede apreciar que el tiempo empleado difiere bastante con el estimado, pero mantiene su estructura y forma, además en el planteamiento inicial es difícil prever ciertos detalles como cuanto o no se solapan las tareas, en un principio se estima tarea a tarea, es decir, una no empieza hasta que la otra termina, esto al final nunca es así, se solapan por necesidad ya que sufren pequeñas modificaciones en base a las primeras experiencias con la práctica.

4.2.2. Lenguajes candidatos.

Cuando se empieza un proyecto de programación una de las primeras preguntas que uno se debería hacer es que lenguaje es más conveniente utilizar. En este caso entre muchos lenguajes conocidos y existentes, fueron candidatos dos de ellos, casi desde el principio, un de ellos fue ***C++*** y el otro ***Python***, pero después de reflexionar sobre las ventajas y desventajas de cada uno, ***Python*** fue el elegido sin ninguna duda.

C++

La elección de ***C++*** como uno de los principales candidatos, reside en una preferencia personal por él, en base a experiencia anterior. Las ventajas que ofrece este sobre ***Python***, es que se puede conseguir mayor eficiencia, aunque no tiene porque ser así, todo depende del programador, también tienes un mayor control sobre todos los recursos, lo que implica que debes dedicar más tiempo y atención a un programa escrito en ***C++*** y esto no siempre es precisamente una ventaja.

Python

Python fue el candidato ganador sin lugar a dudas, por su simplicidad de uso, su productividad y un largo etc…, pero lo que realmente determinó esta decisión fue la cantidad de módulos o librerías escritas en este lenguaje. ***Python*** siempre tiene una solución para todo y en este momento el tiempo premia, debes centrar los recursos disponibles en el objetivo principal y es, en esto, donde ***Python*** resulta mejor opción que ***C++***.

4.2.3. Paradigmas a elegir.

Una vez elegido el lenguaje a utilizar debemos pararnos y ver con más detalle que nos ofrece este mismo, ***Python*** nos brinda la posibilidad de elegir varios paradigmas, entre ellos, la programación orientada a objetos, funcional, imperativa o por procedimientos, en este caso se eligió la programación orientada a objetos debido en gran parte a que el objetivo final es crear una interfaz llena de estos, consiguiendo así una interfaz de aplicación más sencilla.

La programación orientada a objetos es una estructura de almacenamiento y acceso a datos que resulta cómoda e intuitiva, además si añadimos que es uno de los paradigmas

más usados en la actualidad, no había ninguna duda de que este paradigma reunía las mejores cualidades para la fase de desarrollo.

4.2.4. Hardware y Software.

Esta sección detalla el material elegido para la realización de todo el proceso. Primero se expondrá la parte de hardware y más adelante la de software.

Hardware:

Con el fin de aumentar la movilidad, a la hora de trabajar, se ha añadido un ordenador portátil sin ser este imprescindible. Por otro lado el ordenador principal fue más que necesario para el cálculo de las simulaciones, las cuales requieren muchos recursos y trabajar en paralelo, esto hizo que todo este tipo de trabajo recayera sobre él y el portátil también brinda la posibilidad de trabajar en otras tareas, como la memoria, mientras el de escritorio se encarga de realizar simulaciones.

Es necesario tener por lo menos una maquina con un procesador multitarea, también es recomendable que este disponga de tecnología Hyperthreading, lo ideal sería tener más de un equipo de estas características para poder simular varios escenarios en paralelo.

Hardware	Detalles
	**Escritorio genérico Gigabyte** **Placa Base:** Gigabyte GA-H77-DS3H rev: 1.1 **Procesador:** Intel® Core™ i7-3770K Processor (8MB Cache, up to 3.90 GHz) **Memoria:** 2 x 8GB 1600 MHz DDR3 Kingston HyperX **Tarjeta Gráfica:** NVIDIA GeForce 660 GTX 2GB RAM GDDR5 (EVGA) **Pantalla:** Apple HD Cinema Display 30 pulgadas (WQXGA 2560x1600) **Disco:** Seagate 1TB SATA-3 (Máx. 200 MB/s)
	**MacBook Pro (13-inch, Mid 2010)** **Procesador:** 2,66 GHz Intel® Core™ 2 Duo **Memoria:** 8 GB 1067 MHz DDR3 **Tarjeta Gráfica:** NVIDIA GeForce 320M 256 MB **Fusion Drive:** 320 GB SATA2 + 240 GB SSD (Máx. 450 MB/s)

Tabla 5: Hardware utilizado para el desarrollo del trabajo.

Software:

Parte del software venía impuesto por el propio temario del trabajo. El resto fueron elegidos por razones personales y podrían ser remplazados por otros.

Software	Detalles
 **OS X El Capitan**	Sistema Operativo instalado en el ordenador portátil ha sido utilizado en todas las fases, pero la parte que a la memoria se refiere ha sido el encargado de realizar esta tarea por completo, debido a que este equipo no dispone de mucha potencia.
 **Ubuntu 16.04**	Sistema Operativo principal del proyecto, instalado en el ordenador de escritorio el cual contaba con grandes recursos, entre ellos un procesador capaz de atender hasta 8 subprocesos de forma simultánea y un disco de 1 TB, por estas dos razones, fue el encargado de generar los escenarios y calcular las simulaciones.
NetworkX	Este es el framework, escrito en Python, en el que se basa todo el trabajo inicial y las herramientas facilitadas por la Universidad de Alcalá de Henares, este software viene impuesto por necesidad en el trabajo.
 **OMNeT++**	Software impuesto por ser el software de destino, es decir cuando se propuso, OMNeT++ fue elegido para albergar el escenario o red a simular.
 **Anaconda**	Anaconda es el encargado de generar un entorno con todas las dependencias por parte de NetworkX, con él no tenemos que preocuparnos por buscarlas nosotros mismos.
 **Spyder**	Entorno de desarrollo integrado (IDE) para python, dedicado exclusivamente para redes por lo que esta totalmente preparado para trabajar con NetworkX y viene integrado en Anaconda.
 **Photoshop CS5**	Herramienta elegida para trabajar y retocar cualquier tema gráfico, ha sido elegida por ser una herramienta con la que se contaba con experiencia de varios años en el ámbito laboral, además de poseer licencia comercial de este.
 **Pages**	Herramienta de texto utilizada para la maquetación y elaboración de la memoria, elegida por la experiencia anterior además de su agilidad y simplicidad para realizar este tipo de trabajo.
 **Numbers**	Herramienta de hoja de calculo utilizada para el calculo de los resultados obtenidos en simulaciones, elegida por la experiencia anterior además de su agilidad y simplicidad para realizar este tipo de trabajo.

Tabla 6: Software utilizado para el desarrollo del trabajo.

4.2.5. Presupuesto.

Ha continuación se expone el presupuesto estimado para la realización completa del trabajo, en algunos casos el precio es referente a productos semi-nuevos, se incluyen licencias libres a coste cero con el fin de remarcar lo útiles que son estas.

Concepto	Nº	Descripción	Unidad	Total
Salario ingeniero	6	Becario Ingeniero o prácticas Precio por mes bruto.	€ 400	€ 2.400
Escritorio genérico	1	PC + Pantalla HD 30” Precio por unidad semi-nuevo.	€ 1.300	€ 1.300
MacBook Pro 2010	1	MacBook Pro Mediados de 2010 Precio por unidad semi-nuevo.	€ 600	€ 600
Photoshop CS5	6	Software para diseño gráfico, principalmente 2D. Precio por licencia al mes.	€ 60	€ 360
Pages	1	Editor de texto enriquecido, para generar documentación. Precio por licencia.	€ 19	€ 19
Numbers	1	Hojas de calculo, tipo Excel. Precio por licencia.	€ 19	€ 19
OS X El Capitan	1	Sistema operativo Software preinstalado.	€ 0	€ 0
Ubuntu 16.04	1	Sistema operativo Licencia libre.	€ 0	€ 0
NetworkX	1	Framework Python, para estudio de redes. Licencia libre.	€ 0	€ 0
OMNeT++	1	Simulador de eventos discretos, para simular redes. Licencia libre no comercial.	€ 0	€ 0
Spyder	1	Entorno de Desarrollo Integrado (EDI), para Python. Licencia libre.	€ 0	€ 0
			Total:	**€ 4.698**
			IVA:	**€ 987**
			PVP:	**€ 5.685**

Tabla 7: Presupuesto estimado.

4.3. Desarrollo.

El desarrollo es sin duda la parte más importante, ya que sin ella nunca alcanzaríamos el objetivo final. Para explicar todo el proceso se expondrán previamente todas las clases de la versión final para posteriormente nombrarlas según fue necesario añadirlas al proyecto de ***Python***.

Todo desarrollo de aplicaciones o programas se dividen en lo general en una fase con la mínima funcionalidad que recibe el nombre de ***alpha***, otra en la que la funcionalidad es básica que recibe el nombre de ***beta*** y otra en la que la funcionalidad es extendida que recibe el nombre de ***release***, esta se trata de la versión a publicar como primera.

4.3.1. Clases.

En este apartado se expondrá la funcionalidad a grandes rasgos de las clases sin entrar en detalles ni en sus métodos. En cada una de las fases del desarrollo se comentarán en cada caso las modificaciones puntuales que fueron necesarias en ese momento.

Estas clases están orientadas a generar un escenario de simulación para ***OMNeT++*** por lo que su clase principal, que podríamos denominar el núcleo, es una escena y recibe el nombre de ***OSScene***. En esta clase estarán contenidas todas las demás como atributos de esta, cada uno representando un objeto del propio escenario u otros que se encargan de configurar estos.

- **OSScene:** Se trata de la escena o red en cuestión, esta clase es la encargada de leer el grafo y añadir los nodos, almacenar todo la información de estos y aporta métodos para gestionarlos. Se encarga de imprimir toda la configuración de la sección general del archivo de inicialización y el archivo de configuración *IP* en formato *XML*. Fue necesaria desde un principio.
- **OSNode:** Esta es la clase base para crear un nodo ya sea un punto de acceso, cliente o router, existe una clase derivada para los dos primeros. Esta se encarga de guardar la información común de los nodos y generar la *MAC* de los mismos en base a la *ID* que se añade desde ***NetworkX***. Fue necesaria desde un principio.
- **OSAp:** Esta clase es una de las dos que derivan de la clase ***OSNode*** heredando así todos sus atributos y métodos, añadiendo algunos más característicos de este tipo de nodo. Esta clase fue necesaria desde el inicio.
- **OSClient:** Esta clase también deriva de la clase ***OSNode*** y hereda todos sus atributos y métodos, añadiendo algunos más característicos de este tipo de nodo. Esta clase fue necesaria desde el inicio.
- **OSPath:** Esta clase se encarga de gestionar los directorios de trabajo de la aplicación concretamente el de ***OMNeT++*** y en el que tengamos almacenados los archivos de esta misma. Al crear una instancia de objeto solicita estos archivos por medio de un cuadro de diálogo a través del protocolo *X11*. Añadida en fase beta.
- **OSSettings:** Esta clase es la encargada de gestionar y almacenar los pequeños ajustes del escenario y las secciones extras para tráfico *UDP*, *TCP* e *ICMP*. Esta clase fue añadida para la fase release.

4.3.2. Versión Alpha.

Esta versión es la más básica de todas, puesto que aquí el desarrollador lo que busca es obtener la funcionalidad más básica a partir de unas cuantas sentencias de control estudiando así cual es la mejor manera de hacerlo. Esta solo tuvo dos revisiones, la primera buscaba generar los comandos necesarios de ***OMNeT++*** a partir de los datos extraídos del grafo e imprimirlos por pantalla y la segunda simplemente cambiaba la salida por pantalla e imprimía todo esto en archivos de texto. Después de comprobar que los archivos podían ser leídos sin problema por ***OMNeT++***, llegó el momento de empezar con la fase beta y definir algunas de las clases anteriormente expuestas.

```
G = nx.read_gpickle(self.__graph_path + '-NNG.pk')
I = nx.read_gpickle(self.__graph_path + '-UDG.pk')

gc.collect()
wu.initialize(I,G)
print 'Simulación {}'.format(self.__gname)

for i in I.nodes():
  n = str(i)
  if 'listCli' in I.node[i].keys():
    print '*.accessPoint[' + n + '].wlan[0].mac.address = "10:00:00:00:00:00"'
  else:
    print '*.host[' + n + '].wlan[0].mac.address = "10:00:00:00:00:1B"'
```

Captura 23: Código de la primera versión alpha.

```
>>> *.accessPoint[0].wlan[0].mac.address = "10:00:00:00:00:00"
*.host[0].wlan[0].mac.address = "10:00:00:00:00:1B"
*.accessPoint[0].wlan[1].mac.address = "10:00:00:00:00:00"
*.accessPoint[0].wlan[2].mac.address = "10:00:00:00:00:00"
*.host[1].wlan[0].mac.address = "10:00:00:00:00:1B"
*.accessPoint[0].wlan[3].mac.address = "10:00:00:00:00:00"
*.host[2].wlan[0].mac.address = "10:00:00:00:00:1B"
*.host[3].wlan[0].mac.address = "10:00:00:00:00:1B"
...
```

Captura 24: Salida por pantalla de la primera versión alpha.

4.3.3. Versión Beta.

Una vez conseguido todo esto quedaba crear unos archivos lo suficientemente completos como para poder simular en ***OMNeT++*** y es este el objetivo principal de la fase beta. En esta fase se realizaron hasta siete revisiones, implementando en cada una nuevas funciones con el fin de conseguir una versión release.

Revisión beta v0.1

El objetivo de esta revisión consiste básicamente en imprimir en ficheros toda la información extraída del grafo elegido. Para lograr esto hay que leer primero la información de este y almacenarla de una forma estructurada para poder manejar e imprimir de una manera sencilla. El primer paso es generar este archivo con los datos

más básicos del grafo, para poder simular una primera escena, pero esto se consiguió de forma parcial, ya que la configuración de red no estaba presente y fue necesario esperar hasta la siguiente revisión.

```
def __nodePrintToFile(self, name, index, node, outf):
    it = str(index)
    if name == "accessPoint":
        outf.write('*.accessPoint[' + it + '].eth[0].mac.address = "' +
                   node.ethmac + '"\n')
        outf.write('*.accessPoint[' + it + '].wlan[0].mac.address = "' +
                   node.mac + '"\n')
        outf.write('*.accessPoint[' + it + '].wlan[0].radio.channelNumber = '+
                   str(node.channel) +'\n')
    else:
        outf.write('*.' + name + '[' + it + '].wlan[0].mac.address = "' +
                   node.mac + '"\n')
        outf.write('*.' + name + '[' + it +
                  '].wlan[0].mgmt.accessPointAddress = "' + node.ap + '"\n')
        outf.write('*.' + name + '[' + it +
                  '].wlan[0].radio.channelNumber = ‘ +
                   str(node.channel) +'\n')

        if not index % 2:
            outf.write('*.' + name + '[' + it + '].numPingApps = 1\n')
            if not (index == self.__numCl-1):
                outf.write('*.' + name + '[' + it +
                           '].pingApp[0].destAddr = "host['+
                            str(index+1) +']"\n')

    outf.write("*." + name + "[" + it + "].mobility.initialX = " +
                str(node.pos[0]) + "m\n")
    outf.write("*." + name + "[" + it + "].mobility.initialY = " +
                str(node.pos[1]) + "m\n")
```

Captura 25: Método que escribe en el archivo de inicialización los nodos de la red.

Revisión beta v0.2

Esta incorpora un método de la clase ***OSScene*** que se encarga de gestionar la asociación entre clientes y puntos de acceso así como la configuración relativa a la capa 3 del modelo *OSI*, concretamente el protocolo *IP*, para finalmente escribir esto en un fichero *XML*.

```
<config>
      <interface hosts="host[40]" names="wlan0" address="10.0.0.4"
             netmask="255.255.255.x" />
      <route hosts="host[40]" destination="*" gateway="10.0.0.1"
             interface="wlan0"/>

      <interface hosts="host[41]" names="wlan0" address="10.0.0.5"
             netmask="255.255.255.x" />
      <route hosts="host[41]" destination="*" gateway="10.0.0.1"
             interface="wlan0"/>
...
```

Captura 26: Archivo XML que contiene la configuración IP.

```
def __assoApClient(self):
    print 'Asociando clientes con puntos de acceso ...'
    for node in self.__Aps:
        indexAp = self.__Aps.index(node)

        for client in node.listCli:
            indexCl = self.__index[str(client)]
            self.__Cls[indexCl].ap = node.mac
            self.__Cls[indexCl].channel = node.channel
            self.__xml.write('\t<interface hosts="host[' +
                             str(self.__Cls.index(self.__Cls[indexCl])) +
                             ']" names="wlan0" address="10.' +
                             str(indexAp) + '.0.' +
                             str(node.listCli.index(client)+2) +
                             '" netmask="255.255.255.x" />\n')

            self.__xml.write('\t<route hosts="host[' +
                             str(self.__Cls.index(self.__Cls[indexCl])) +
                             ']" destination="*" gateway="10.' +
                             str(indexAp) + '.0.1" interface="wlan0"/>\n\n')
```

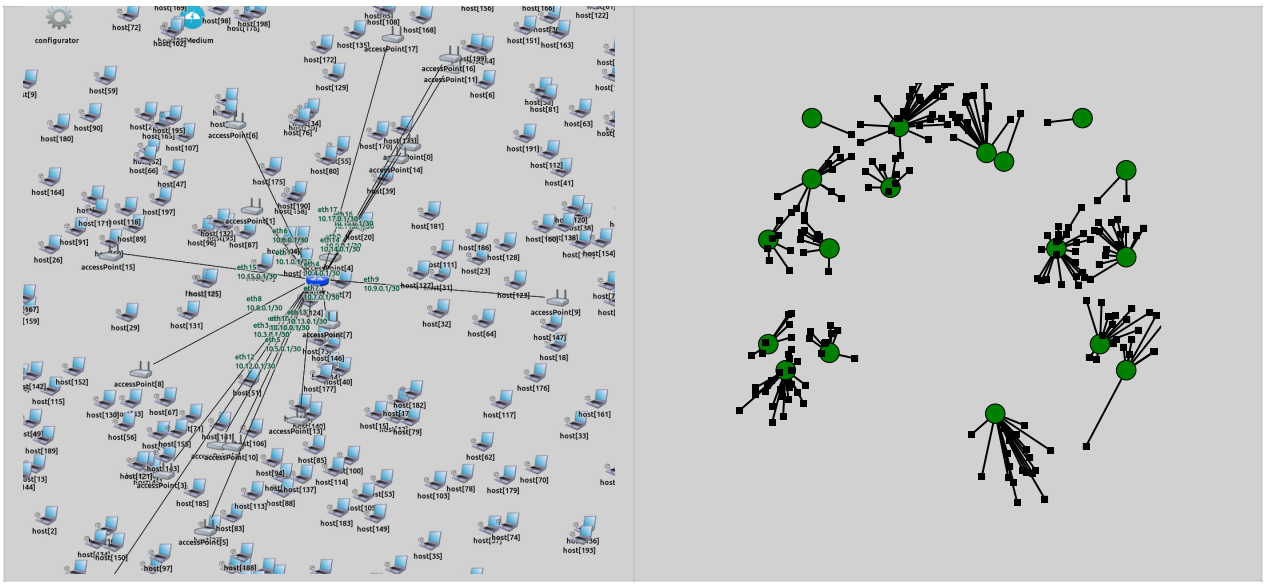
Captura 27: Método encargado del proceso de asociación.

Revisión beta v0.3

Esta incorpora una serie de plantillas para facilitar la generación de archivos necesarios y copiarlos en el proyecto de ***OMNeT++***. Esto generó la necesidad de gestionar varios directorios, concretamente este mismo y el de la aplicación ***OGSimulation***, fue así como se decidió añadir la clase ***OSPath***, que gestiona todo este problema. Llegados a este punto teníamos ya un escenario a simular, pero todavía algo estaba fallando ya que las posiciones eran las correctas y parecía que ***OMNeT++*** no hacia caso alguno a ellas, pero estaban escritas y bien asignadas.

```
class OSPath(object):
    def __init__(self, opp_path = "", ogs_path = ""):
        if opp_path == "" or ogs_path == "":
            root = tk.Tk()
            root.withdraw()
            opts = {}
            opts['title'] = "Directorio del proyecto OMNet++"
            self.opp = tkFileDialog.askdirectory(**opts) + "/"
            opts['title'] = "Directorio de OGSimulation"
            self.ogs = tkFileDialog.askdirectory(**opts) + "/"
        else:
            self.opp = opp_path
            self.ogs = ogs_path

    def getOPPSimulationPath(self):
        return self.opp + "simulations/"

    def getOPPImagesPath(self):
        return self.opp + "images/"

    def getOGSTemplatesPath(self):
        return self.ogs + "templates/"

    def getOGSImagesPath(self):
        return self.ogs + "images/"
```

Captura 28: Código de la clase completa OSPath.



Captura 29: Comparando escenario generado con beta v0.3 con el grafo de NetworkX.

En el escenario de la revisión v0.3 se puede apreciar que no representa las posiciones en las que deben aparecer los distintos nodos de la red.

Revisión beta v0.4

Esta revisión su meta principal consiste en corregir las posiciones erróneas, pese a que la solución fue bastante fácil, llevó bastante tiempo dar con la solución, que consiste en añadir un simple comando en el archivo de inicialización de manera que ahora sí que ***OMNeT++*** hacía caso a las posiciones que se habían asignado, pero surgieron varios errores al contener algunas de estas con valor negativo.

```
**.mobility.initFromDisplayString = false
```

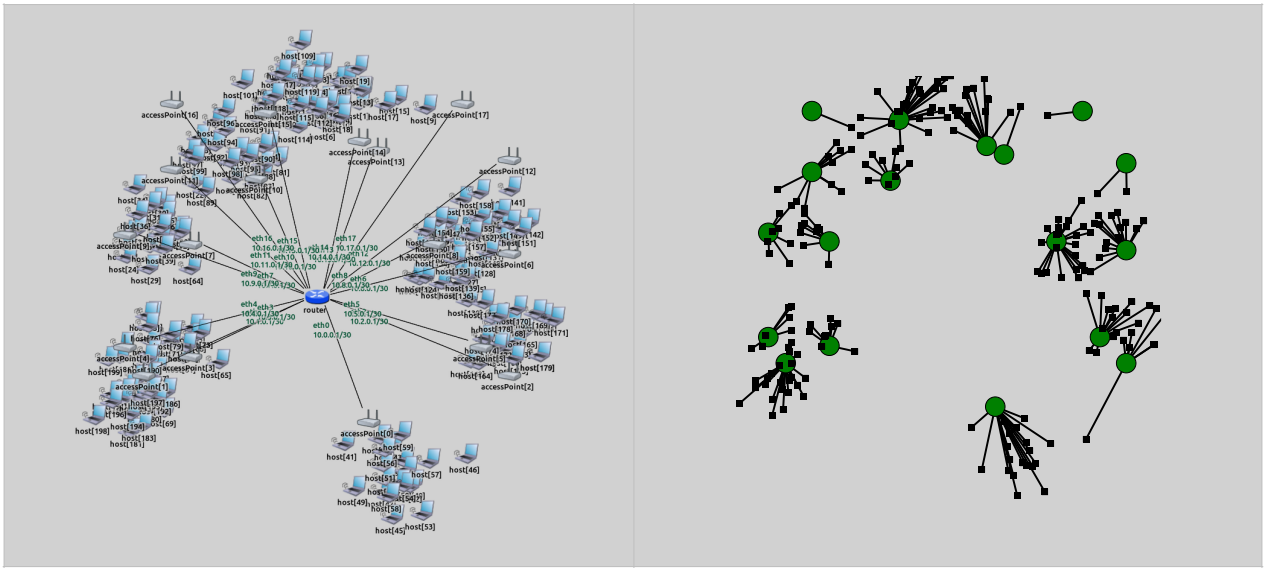
Captura 30: Comando que resuelve el problema de posiciones erróneas.

Revisión beta v0.5

En la anterior revisión surgió un imprevisto con las posiciones negativas y es la v0.5 la encargada de resolver este problema, detectando estas y añadiendo un offset si fuera necesario.

```
def __fixPosition(self):
  for acp in self.__Aps:
      acp.pos[0] = acp.pos[0] + abs(self.__Xmin * 100)
      acp.pos[1] = acp.pos[1] + abs(self.__Ymin * 100)

  for cli in self.__Cls:
      cli.pos[1] = cli.pos[1] + abs(self.__Ymin * 100)
      cli.pos[1] = cli.pos[1] + abs(self.__Ymin * 100)

  print 'Corregidas posiciones negativas.'
```

Captura 31: Método encargado de corregir posiciones negativas.



Captura 32: Comparando escenario generado con beta v0.5 con el grafo de NetworkX.

En este último escenario ya se puede apreciar la forma del grafo, habiendo así alcanzado nuestro objetivo de la forma más básica, porque el coloreado de este aún no cumple con el experimento a realizar, ya que esto se genera de forma totalmente aleatoria.

Revisión beta v0.6

En este momento se incorporó la posibilidad de elegir el coloreado de una forma muy sencilla, este se pasa como argumento directamente en un array y se asigna en el mismo orden que leemos los puntos de acceso, es decir, en el mismo orden que los añadimos a la escena. De esta manera se le da mayor versatilidad y dinamismo a la hora de probar distintos coloreados.

```
scene = ogs.OSScene(graph_name, [4, 7, 3, 7, 2, 10, 2, 2, 11, 6], 'EXP1')
```

Captura 33: Pasando coloreado como argumento al iniciar la escena.

```
if self.__rand:
    color = randint(1,11)
    self.__color.append(color)
else:
    color = self.__color[self.__numAp]
```

Captura 34: Código del coloreado en función del array pasado como argumento.

Revisión beta v0.7

Esta sólo incorpora una mejora visual respecto a su anterior versión. La cual consiste en añadir iconos de equipos más modernos de los que ofrece ***OMNeT++*** por defecto y se ajusto como fondo el plano de la Escuela Politécnica Superior, con todo esto, esta versión fue la candidata para la versión release.

		
iPhone 6	**MacBook Pro**	**MacBook**
		
iMac 27”	**Wireless Router**	**AirPort**
		
Samsung Galaxy S7	**Radio Medium**	**Settings**

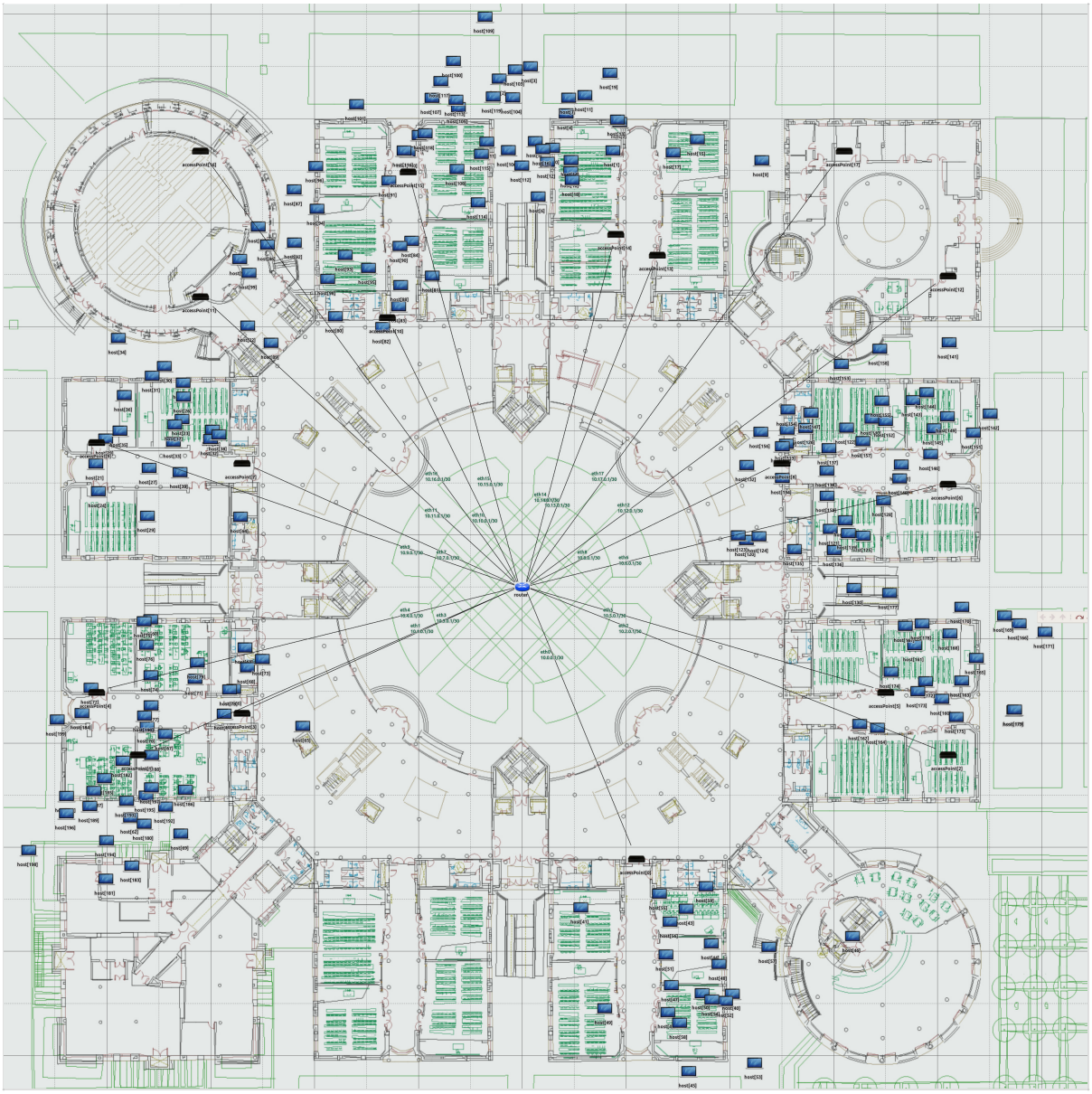
Tabla 8: Algunos iconos de los añadidos en la revisión v0.7.

4.3.4. Versión Release.

En la versión release se realizó la última revisión y se incluyó la última clase ***OSSettings***. Esta se encarga de guardar todos los ajustes de interfaces tanto ***Wifi*** como *Ethernet* así como la configuración de servidores y clientes, tanto para *TCP*, *UDP* e *ICMP*. En este tipo de objeto se delega todo lo relacionado con configuraciones, cada interfaz tiene una serie de parámetros para personalizar esta, que se ven en detalle en la sección de referencia de esta misma memoria, en el caso de aplicaciones también pueden ser personalizadas desde la longitud del paquete como los hosts que actúan como cliente o servidor. Estas últimas modificaciones fueron necesarias para poder simular algo de tráfico en el escenario generado.

```
*.host[0].numUdpApps = 1
*.host[0].udpApp[*].typename = "UDPVideoStreamSvr"
*.host[0].udpApp[*].sendInterval = 10ms
*.host[0].udpApp[*].packetLen = 1000B
*.host[0].udpApp[*].videoSize = 1000MiB
*.host[0].udpApp[*].localPort = 3099

*.host[*].numUdpApps = 1
*.host[*].udpApp[*].typename = "UDPVideoStreamCli"
*.host[*].udpApp[*].serverAddress = "host[0]"
*.host[*].udpApp[*].localPort = 9999
*.host[*].udpApp[*].serverPort = 3000
*.host[*].udpApp[*].startTime = 0
```

Captura 35: Configuración de video en streaming UDP, un servidor y el resto clientes.



Captura 36: Escena generada pos OGSimulation de la Escuela Politécnica Superior.

4.4. Análisis de resultados.

A la hora de analizar resultados hay que elegir muy bien el escenario más conveniente en cada caso, las simulaciones requieren mucho tiempo de ejecución y este es un recurso limitado siempre, por lo que hay que definir el escenario de la manera más eficiente, con el fin de obtener resultados en el menor tiempo posible.

4.4.1. Configuración del escenario.

Para poder sacar conclusiones fiables, debemos repetir el escenario un mínimo de 10 veces, lo ideal serían a partir de 20, pero como se ha indicado anteriormente las simulaciones requieren mucho tiempo de ejecución, por lo que hay que definir el escenario lo más sencillo posible y que cumpla con el objetivo.

En el caso que nos ocupa, se determino que el tráfico tiene que ser unidireccional, la red tiene que estar lo más cargada posible y los equipos involucrados han de ser los mínimos con el fin de agilizar la simulación. Con estas conclusiones se determino el escenario que desglosa la tabla 9.

Clientes UDP	10
Servidores UDP	10
Tiempo (s)	15
Repeticiones	10

Tabla 9: Datos del escenario a simular.

Se escogieron aquellos hosts más conflictivos, es decir, aquellos que se encuentran dentro del radio de cobertura de dos puntos de acceso diferentes.

La configuración en ***OGSimulation*** se realizó de la siguiente manera:

Para todo el conjunto de pruebas, se ha escogido como escenario la planta baja de la Escuela Politécnica Superior de la Universidad de Alcala de Henares. Con 26 nodos como puntos de acceso, que reciben coordenadas según su posición real, ajustando estas al plano CAD de la Escuela Politécnica Superior. Por otro lado tenemos, nodos que actúan como clientes, los cuales fueron repartidos siguiendo una distribución normal, con media en el centro de las aulas y desviación estándar normalizada al tamaño del escenario de 0.05. Se ha estimado una ocupación media de 20 alumnos por clase, con un total de 48 de estas. Para este experimento se ha escogido una ocupación del edificio de manera que únicamente el 20% de las aulas están siendo ocupadas, lo que deja un total de 10 aulas ocupadas y 200 clientes en el escenario. Aquellos puntos de acceso que se encuentran en estado inactivo, al no tener ningún cliente asociado, han sido eliminados de este experimento, por lo que no se tendrán en cuenta en la simulación y tampoco serán representados, motivo por el cual han quedado 18 puntos de acceso.

Han sido elegidos tres escenarios en concreto para este estudio. El primero que vamos a nombrar, se ha denominado el peor caso y ha sido bautizado como *WORST*, debido a su significado en inglés. Este es el más simple de todos, consiste en asignar el mismo canal a todos los puntos de acceso, con el fin de que sea el peor puntuado, sirviendo así de referencia para los demás. En otro de los escenarios, se ha seleccionado el coloreado mediante el mecanismo de decisión *SA* que ha sido visto, anteriormente, en el apartado 2.2.3, de esta memoria. En el último de estos tres escenarios, el coloreado lo define un mecanismo conocido como *LCCS (Least Congested Channel Search)*, este no fue elegido por casualidad, ya que es usado por defecto en el protocolo ***IEEE 802.11***.

```
scene.set.wlan['opMode'] = 'n'
scene.set.wlan['bandwidth'] = '22MHz'
scene.set.wlan['txPower'] = '30mW'
scene.set.wlan['sensitivity'] = '-90dBm'
scene.set.wlan['gain'] = '10dB'

scene.set.udp['videoSize'] = '1000MiB'
scene.set.udp['sendInterval'] = '10ms'
scene.set.udp['packetLen'] = '1000B'

scene.set.udp['servers'] = range(49,59)
scene.set.udp['clients'] = range(109,119)

scene.set.ping['background'] = True
scene.set.ping['packetSize'] = '1024B'
scene.set.ping['printPing'] = True
```

Captura 37: Configuración para la simulación final.

Se han ajustado los parámetros de todas las interfaces ***Wifi*** como se ha expuesto en el apartado 4.1.1 se han escogido como servidor, los que se encuentran en una zona despejada de interferencias y los clientes en una zona conflictiva, en la que existen varios puntos de acceso en sus proximidades. En la captura 37, muestra una parte de todos estos ajustes.

4.4.2. Analizando los datos obtenidos.

Después de simular este experimento, a una media de un escenario por hora, se obtuvieron los resultados para su análisis. Se puede comprobar que las pérdidas no son demasiadas en ninguno de los casos, pero para sacar las primeras conclusiones pueden ser suficientes.

Aquellos hosts que actúan como servidor *UDP* que no han llegado a enviar nada, debido a las interferencias, han sido eliminados para el cálculo del valor medio, pero es un dato que deberíamos tener en cuenta puesto que en algunos casos han logrado enviar muy pocos y esto puede dar lugar a que los pocos que llegan a conseguirlo no encuentren muchos problemas, al no haber otros hosts emitiendo tráfico, por lo que se obtienen muy buenos resultados frente a pérdidas. Pero la realidad es algo diferente, puesto que no han podido enviarse muchos de estos paquetes, por lo que se debería realizar un ranking teniendo en cuenta esto, para obtener conclusiones más precisas.

UDP | 10 SERVER 10 CLIENTS | 18 AP 200 HOST

Repetición	WORST	LCCS	SA
0	7,60%	4,38%	3,01%
1	8,08%	3,04%	2,42%
2	4,74%	2,83%	1,99%
3	4,76%	4,43%	2,95%
4	5,46%	2,40%	2,44%
5	8,45%	4,73%	2,75%
6	6,52%	4,15%	1,93%
7	10,03%	3,99%	3,28%
8	8,09%	4,79%	3,72%
9	6,21%	1,84%	3,04%
min	**4,74%**	**1,84%**	**1,93%**
max	**10,03%**	**4,79%**	**3,72%**
STD	**1,74%**	**1,05%**	**0,56%**
AVG	**6,99%**	**3,66%**	**2,75%**

Tabla 10: Resultado de pérdidas UDP en los escenarios simulados.

Para verificar la fiabilidad de los resultados debemos obtener los intervalos de confianza. Esto se puede calcular, conociendo la desviación típica y el número de repeticiones. En estadística se conoce como intervalo de confianza a un par de valores o conjunto de pares, en el que se estima encontrar el valor de una variable aleatoria, con una probabilidad de acierto determinada, esta probabilidad se representa como (1 - α). Para

el caso que nos ocupa se ha determinado un α de 0.05, es decir una probabilidad de acierto de un 95% y los resultados, para el porcentaje de paquetes perdidos, son los siguientes:

LCCS (%)	$3,66 \pm 0,75$
SA (%)	$2,75 \pm 0,40$
WORST (%)	$6,99 \pm 1,24$

Tabla 11: Intervalos de confianza.

4.5. Conclusiones.

Todo el desarrollo del trabajo, ha sido una labor entretenida e interesante. Al empezar daba la impresión de ser más simple de lo que en realidad ha resultado. Ha requerido mucho más tiempo y esfuerzo del que se pudo estimar en un principio, pero en todos los aspectos ha sido una experiencia enriquecedora e interesante.

Los resultados obtenidos en las simulaciones han sido muy similares a los esperados de experimentos anteriores y cálculos teóricos. No se disponía del tiempo suficiente para poder simular escenarios de mayor duración y mayor número de repeticiones, por lo que las conclusiones obtenidas, son buenas, pero puede que sean insuficientes. Ahora queda en manos del equipo que ha seguido este trabajo, el investigar con ello, para sacar conclusiones más precisas. Es cierto, que este no era el objetivo principal del trabajo, simplemente se consideró que podía ser interesante hacer una prueba sobre varios escenarios generados con la aplicación desarrollada. Muchas de estas simulaciones fueron necesarias para poder sacar conclusiones a la hora de determinar como generar el escenario, de la manera más fiel al modelo analítico en que se basa este trabajo.

Todo este desarrollo brinda la oportunidad al estudiante de hacer uso de algunos de los muchos conocimientos adquiridos a lo largo de toda la carrera. De estos conocimientos, a parte de todos los datos técnicos que ha requerido el desarrollo de este trabajo, cabe destacar la capacidad para aprender e investigar. Quizá esta es la cualidad más importante que uno adquiere en este tipo de estudios, el no rendirse y siempre buscar más allá.

En este caso la información era escasa ya que ***OMNeT++*** no es un software usado a gran escala y la documentación y ejemplos brillaban por su ausencia. Cada prueba que se realizaba requería varias horas de simulación y posteriormente el análisis de los resultados. Esto ha sido una dificultad bastante importante en todo el desarrollo.

A título personal ha sido muy interesante poder aplicar estos conocimientos, me ha ayudado a ser consciente de todo lo aprendido a lo largo de todos estos años de estudio y ver que uno puede ser capaz de hacer cosas que antes de empezar, todo este camino, hubieran resultado francamente difíciles.

5. MANUAL DE USUARIO.

Esta sección está destinada a describir la forma correcta de uso para esta aplicación, desde el proceso de instalación hasta cómo gestionar y analizar los resultados del escenario final simulado en ***OMNeT++***.

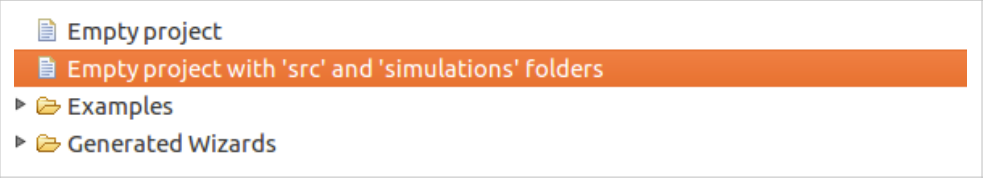
5.1. Instalación.

El proceso de instalación es tan sencillo que casi se podría decir que no existe, lo único necesario es copiar y pegar la carpeta raíz de ***OGSimulation*** en la ubicación deseada, una vez hecho esto, simplemente hay que generar un nuevo script e indicar el directorio raíz de la aplicación y la ubicación de las dependencias que se indican más adelante.

```
sys.path.append('../OGSimulation')  # Directorio raíz de OGSimulation.
sys.path.append('../Grafos-v2')     # Directorio de Utilidades Uah.
import OSScene as OGS
```

Captura 38: Instalación de OGSimulation.

Antes de seguir debemos crear un proyecto vacío de ***OMNeT++***, navegamos hasta *File > New > OMNeT++ Project …* y elegimos la opción proyecto vacío con directorios "*src*" y "*simulations*", por último es necesario añadir como referencia el framework ***INET***.



Captura 39: Creando proyecto vacío con directorios src y simulations.

5.1.1. Dependencias.

A continuación se expone una tabla con las dependencias de software que requiere para un correcto funcionamiento, indicando la versión para la cual fueron probadas.

Dependencia	Descripción	Versión
Spyder	Provee el entorno de librerías que son necesarias para ejecutar la aplicación.	2.3.9
NetworkX	Librería de redes basadas en técnicas de teoría de grafos.	1.11
OMNeT++	Software destino para poder simular los resultados.	5.0
INET	Framework para OMNeT++, contiene modelos basados en los estándares de IEEE.	3.2.4-b32e1dc
Ubuntu	La aplicación es sólo compatible con este sistema operativo.	14.04 LTS+

Dependencia	Descripción	Versión
wifigraphs	Utilidad facilitada por el departamento de automática de la Universidad de Alcalá de Henares.	2.0
wifigraphs_parameters	Utilidad facilitada por el departamento de automática de la Universidad de Alcalá de Henares.	2.0
wifi_utility	Utilidad facilitada por el departamento de automática de la Universidad de Alcalá de Henares.	2.0

Tabla 12: Dependencias.

5.2. Creando una escena.

Para crear una escena sólo necesitamos pasarle tres parámetros, de los cuales dos son opcionales. El primero y el único obligatorio sería la ruta del grafo. El segundo debe contener el coloreado en cuestión. Lo ideal es que este array tenga la misma longitud que el número de puntos de acceso, si no es así, en el caso de ser de menor longitud, aquellos puntos de acceso que se queden sin asignar, se generará un color aleatorio en cada caso. De no pasar ningún array todos los puntos de acceso recibirán un color aleatorio. Por otro lado el tercer parámetro es simplemente un sufijo para nombrar el proyecto y así poder distinguirlo frente a otra configuración del mismo escenario, como es opcional si no se le da ningún valor no tendrá ningún impacto sobre dicho nombre.

```
name = 'EPS-0.2-1_1'
prename = [‘20160607sa3000t1_eps’,
           '20160607opt2','20160607hc3000_eps',‘20160727lccs']

cselec = ['SA','ALPSO','HC','LCCS10_T1_P2','RAND','WORST']

for i in [0,3,4,5]:
    if i < 4:
        salida = open('results/{}/{}_{}_{}'.format(prename[i],
                                                   prename[i],
                                                   name,
                                                   cselec[i]), 'rb')


        for j in range(0,7):
            experiment = pickle.load(salida)

        cselec[i] = cselec[i] + '_' + str(j + 1)
        color = experiment['color']

    else:
        if cselec[i] == 'RAND':
            color =  []
        else:
            color = [6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6]


    graph_name = './grafoEPS/{}'.format(name)

    scene = ogs.OSScene(graph_name, color, cselec[i])
```

Captura 40: Creando varias escenas con distintos coloreados.

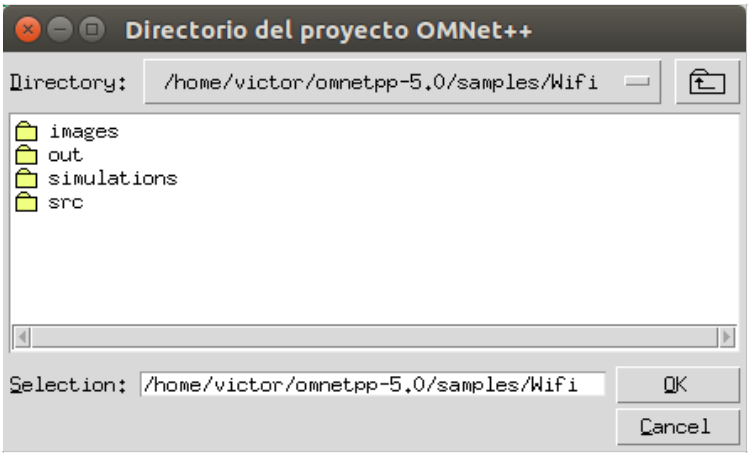
En la captura 40 se puede apreciar como se crean tres escenarios *SA*, *LCCS* y *WORST*, mediante un bucle for, esto es sólo un ejemplo de como lanzar escenas diferentes de forma automatizada. El resultado en ***OMNeT++*** sería un conjunto de carpetas con los diferentes nombres de coloreado como refleja la captura 41.

El proyecto de ***OMNeT++*** no sufrirá ningún cambio hasta el momento que se lance la escena, esto es algo que se expondrá más adelante.

Captura 41: Organización de escenarios en directorios.

5.2.1. Directorios.

Los directorios de trabajo se solicitan al usuario antes de empezar a leer el grafo, por medio del protocolo *X11,* que muestra un cuadro de diálogo en el que el usuario puede navegar hasta el directorio deseado. Primero se solicita el directorio que contiene el proyecto de ***OMNeT++*** y el segundo donde se encuentra la carpeta con la aplicación ***OGSimulation***.



Captura 42: Solicitando directorio del proyecto destino para OMNeT++.

5.2.2. Personalizando el escenario.

OMNeT++ te permite ajustar muchos parámetros para la simulación, pero se han portado solamente los que se han considerado más relevantes para el caso que nos ocupa. Estos ajustes tienen que ver con la configuración de las interfaces y aplicaciones para generar tráfico. En la captura 43 podemos ver un ejemplo de configuración de interfaces Ethernet y ***Wifi***, así como activar el tráfico de fondo, el tiempo de simulación o el número de repeticiones que queremos realizar. Todos estos ajustes se describen con detalle en la sección 6 destinada a la referencia de uso.

```
scene.set.eth['promiscuous'] = False
scene.set.eth['duplexMode'] = True
scene.set.eth['mtu'] = '1500B'

scene.set.wlan['opMode'] = 'n'
scene.set.wlan['bandwidth'] = '20MHz'
scene.set.wlan['txPower'] = '30mW'
scene.set.wlan['sensitivity'] = '-90dBm'
scene.set.wlan['gain'] = '10dB'

scene.set.ping['background'] = True
scene.set.ping['packetSize'] = '252B'
scene.set.ping['printPing'] = True

scene.set.timeLimit = '30s'
scene.set.repeat = 10
```

Captura 43: Configuración de interfaces y tráfico de fondo.

Hay que destacar que para las aplicaciones *TCP* y *UDP* se pueden determinar los hosts que actúan como clientes y los que lo hacen de servidores. La asignación se hace de forma equitativa y secuencial, es decir que si tenemos cinco clientes y cinco servidores el primer cliente se comunicará con el primer servidor y así sucesivamente en el caso de que el número de clientes sea el doble que el de servidores los dos primeros clientes se asignarán al primer servidor y así sucesivamente, de esta manera podemos configurar distintos perfiles de tráfico, incluso elegir un servidor central para todos los clientes.

```
scene.set.udp['videoSize'] = '1000MiB'
scene.set.udp['sendInterval'] = '10ms'
scene.set.udp['packetLen'] = '1000B'

scene.set.udp['servers'] = range(49,59)
scene.set.udp['clients'] = range(109,119)

scene.set.tcp['active'] = True
scene.set.tcp['active'] = '1000MiB'

scene.set.tcp['servers'] = [4,9,115,135,41,198,24,101,6,141]
scene.set.tcp['clients'] = [164,53,69,27,96,3,149,171,49,199]
```

Captura 44: Configuración de aplicaciones TCP y UDP.

5.2.3. Lanzando la escena.

Una vez creada y configurada la escena, sólo queda un paso, lanzar la escena para que se generen todos los archivos y directorios necesarios para simular nuestros escenarios y así poder sacar conclusiones. Una vez generado el proyecto debemos compilarlo pulsando con el botón derecho sobre el nombre de este y hacemos click en la opción *Build Project*.

```
scene.launchScene()
```

Captura 45: Lanzando la escena.

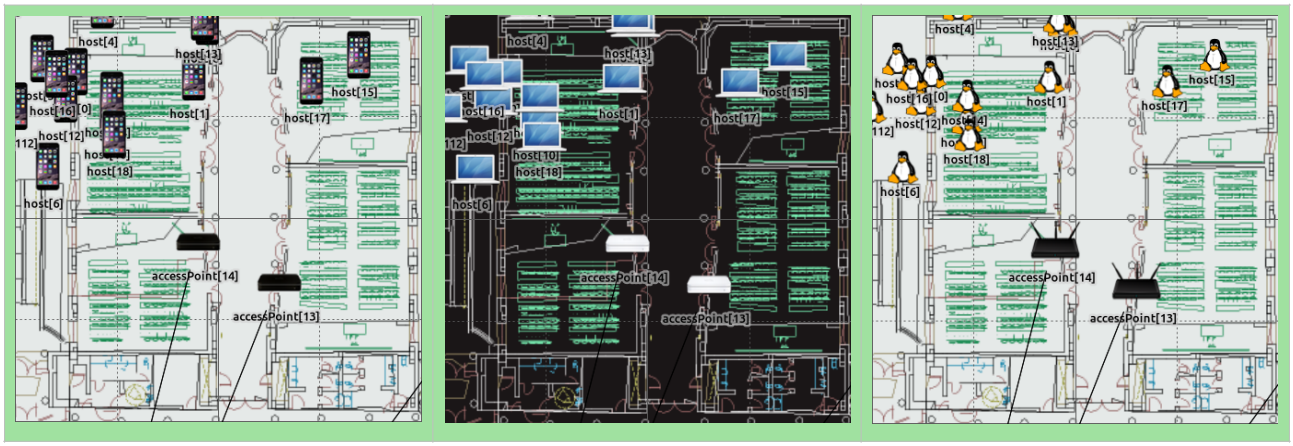
5.3. Simulando con *OMNeT++*.

Una vez hayamos lanzado la escena se crean todos los archivos "*.ini*" de cada escenario, así como un "*.xml*" con la configuración de red *IPv4*, estos tendrán el nombre del escenario y se encontrarán dentro de una carpeta con este mismo nombre, dentro del directorio "*simulations*" donde también se crearán los siguientes ficheros:

Graph80211.ned	Archivo de interfaz gráfica del escenario a simular.
omnetpp_test.ini	Archivo de inicialización simplificado, escenario simple para pequeñas pruebas.
network.xml	Archivo de configuración IPv4 para el escenario simplificado.

Tabla 13: Archivos globales generados.

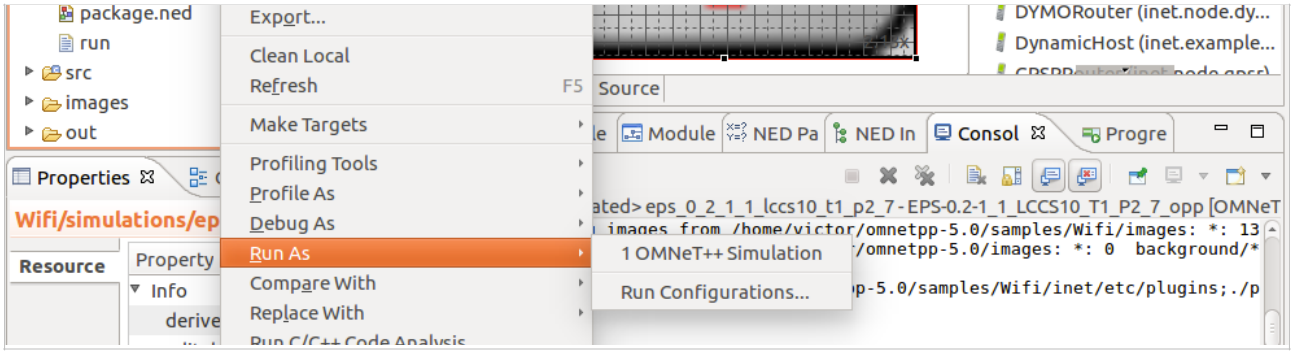
Para poder personalizar el escenario se añade al proyecto de ***OMNeT++*** una carpeta con imágenes de la planta baja de la Escuela Politécnica Superior que usaremos para incrustar de fondo situando así los puntos de acceso en los lugares correspondientes, también contiene iconos para los hosts o puntos de acceso.



Captura 46: Ejemplos de personalización gráfica.

5.3.1. Simulación.

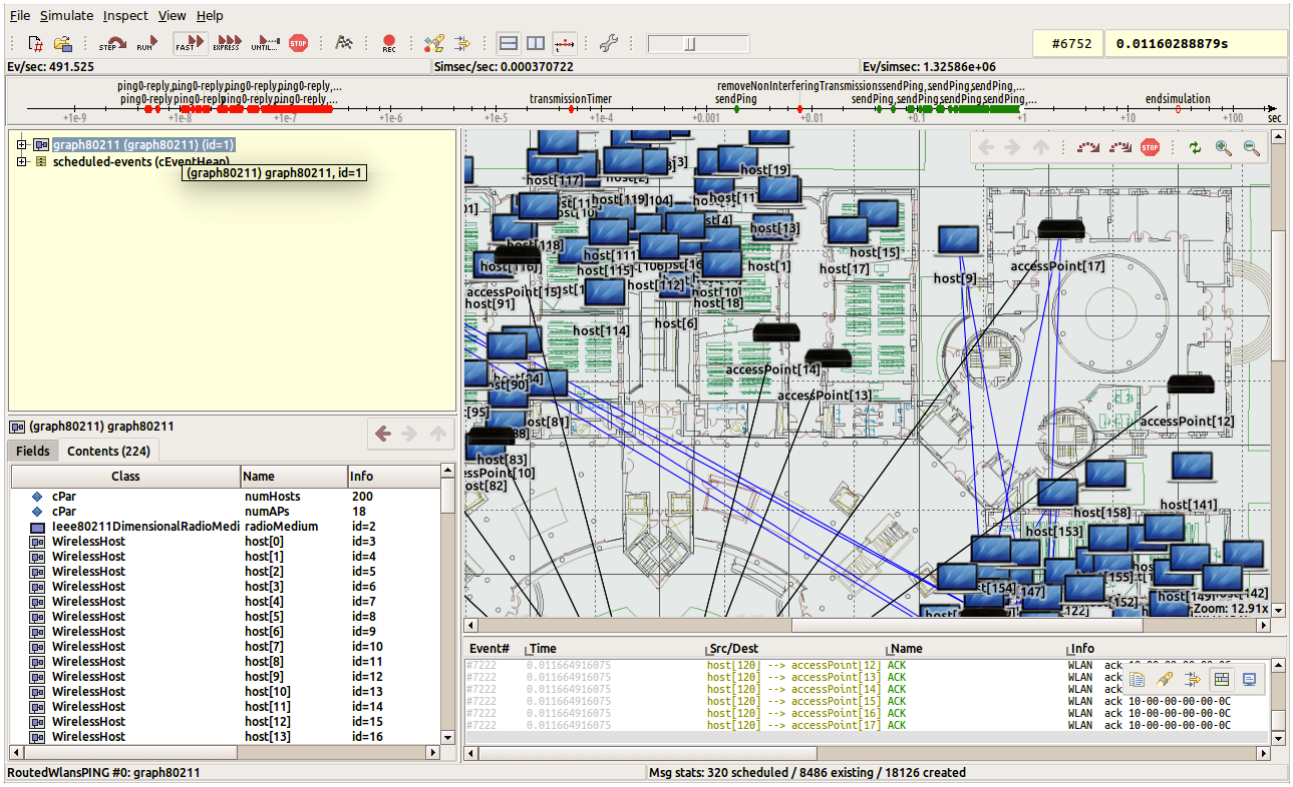
Para simular en ***OMNeT++*** basta con seleccionar el archivo "*.ini*" en cuestión, pulsar el botón derecho del ratón sobre él y desplazarnos al menú *Run as > OMNeT++ Simulation*.



Captura 47: Iniciando la simulación del escenario generado.

Esto lanzará una nueva ventana con el escenario listo para simular. Para iniciar la simulación tenemos varias opciones todas ellas en el menú 

que podemos encontrar arriba a la izquierda. En este menú podemos distinguir cuatro tipos de velocidad para la simulación y los dos botones restantes son para determinar un instante de parada el primero y el último detendría la simulación en un momento determinado por el usuario.

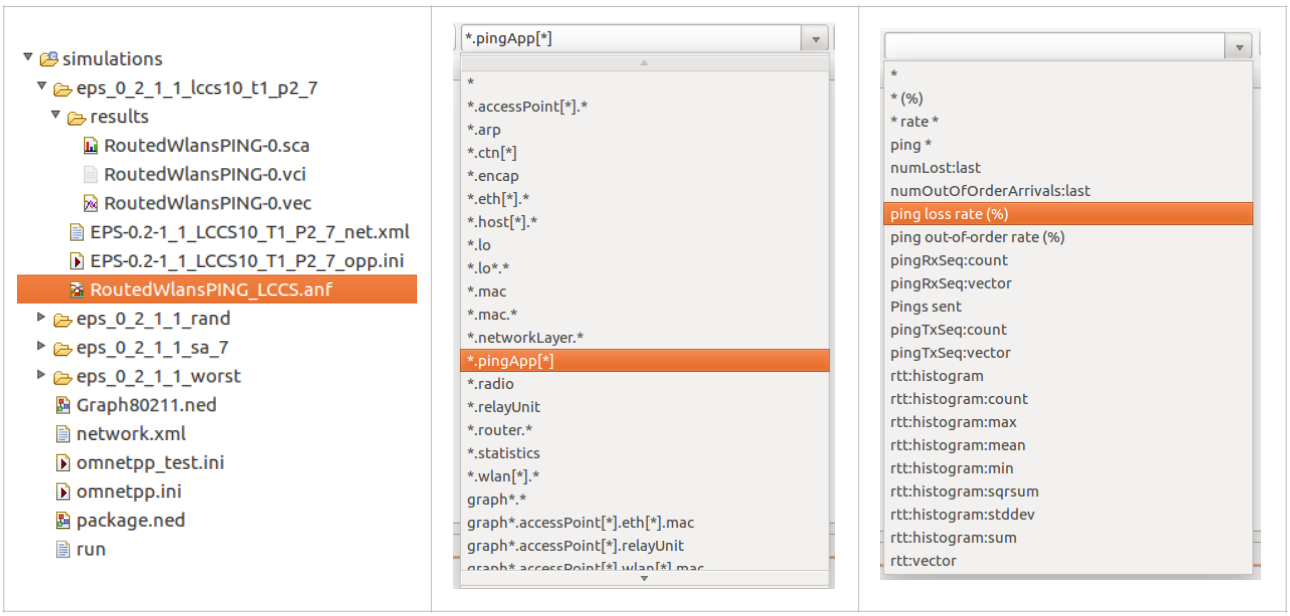


Captura 48: Corriendo la simulación del escenario generado.

5.3.2. Obtener datos.

Para manejar los datos que genera la simulación, ***OMNeT++*** nos ofrece un tipo de archivo con extension *".anf"* que gestiona todos estos.

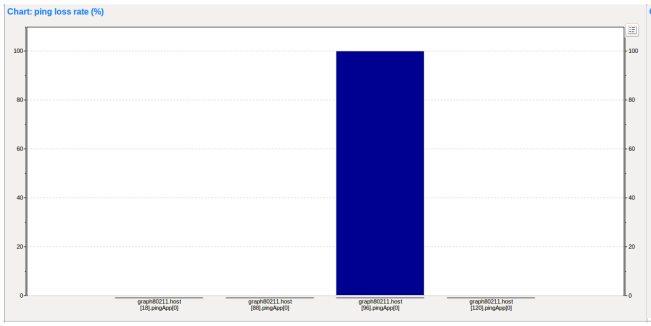
Al terminar la simulación se crea una carpeta, con el nombre *"results",* por ejecución. Esta contiene tres archivos con el mismo nombre que la sección, del archivo de inicialización, elegida para simular. Haciendo doble click con el botón principal del ratón, en cualquiera de ellos, automáticamente se crea un archivo con la extensión *".anf"* con el mismo nombre. Al abrir este archivo nos encontramos con varias pestañas. De momento la que vamos a prestar atención es "*Browse Data*". Al dirigirnos a esta nos muestra tres desplegables, el primero es simplemente para seleccionar una ejecución concreta, el siguiente filtra el módulo que queremos y el último el tipo de dato. Una vez seleccionado el filtro deseado sólo se mostrarán aquellos nodos que tengan algún submódulo que coincida con este tipo de datos, es decir, si seleccionamos aquellos módulos que tengan aplicaciones *UDP* sólo mostrará los que contengan este tipo de submódulo y si dentro de estos volvemos a filtrar por tipo de datos, por ejemplo paquetes enviados, sólo se mostrarán aquellos que han jugado el papel de servidor. A continuación un ejemplo filtrando las pérdidas que se han producido en las aplicaciones *PING* o *ICMP*.



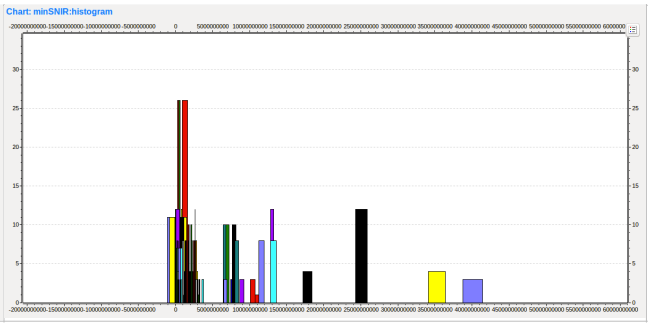
Captura 49: Proceso resumido de obtención de datos.

Estos datos pueden ser exportados en dos formatos, "*.csv*" archivos delimitados por comas o "*.octave*" que es un formato de texto que se puede abrir con varios editores de texto.

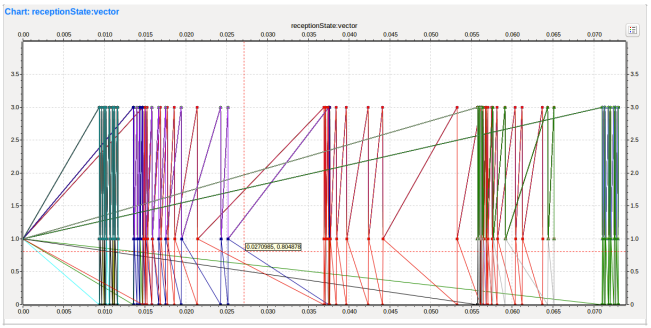
Todos estos datos pueden ser representados gráficamente, de hecho se dividen en tres grupos: escalares, vectores e histogramas. Estos pueden ser representados de una forma muy versátil, seleccionando aquellos módulos que nos interesan, además de personalizar a nuestro gusto las gráficas de resultados.



Captura 50: Gráfica de tipo escalar.



Captura 51: Gráfica de tipo histograma.

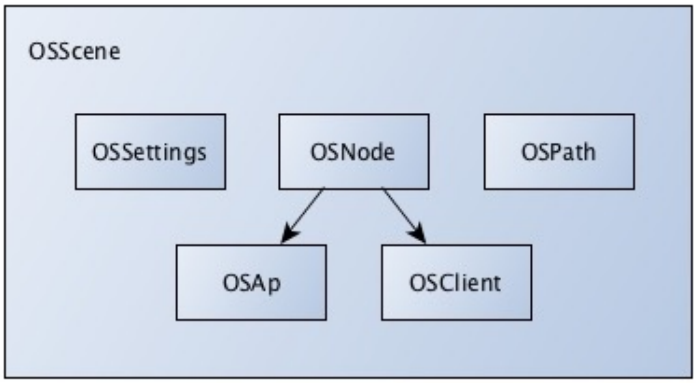


Captura 52: Gráfica de tipo vector.

6. REFERENCIA.

En esta sección se describen los atributos y métodos que contienen las distintas clases de la aplicación, se detallan los tipos de datos que almacenan o que reciben como parámetro, en el caso de los métodos, así como sus valores por defecto. Cuando algún dato deba ser introducido de forma explícita se indicará cual es su valor por defecto con un asterisco (*).

La jerarquía de esta aplicación es muy simple. Básicamente existe una clase principal que contiene a todas las demás como atributos de esta. Existe un sólo caso de herencia en el que dos clases heredan de una clase base. A derecha se representa esta jerarquía en un diagrama.



6.1. Clase OSScene.

```
class OSScene(object):
```

Esta es la clase principal que contiene a todas las demás, las cuales son atributos de la misma, por lo que podríamos decir que esta clase es el núcleo de la aplicación. Esta consta de varios atributos y métodos que explicaremos a continuación.

6.1.1. Atributos:

La mayoría de los atributos son privados, esto se debe a que la clase va a trabajar con ellos de forma interna sin necesidad de ser accesibles.

Atributo	Tipo	Default	Función
`self.__Cls`	`[OSClient]`	`[]`	Contiene los objetos de la clase OSClient que representan los clientes.
`self.__numCl`	`Int`	`0`	Número de clientes.
`self.__Aps`	`[OSAp]`	`[]`	Contiene los objetos de la clase OSAp que representan los puntos de acceso.
`self.__numAp`	`Int`	`0`	Número de puntos de acceso.
`self.__index`	`{Int}`	`{}`	Mapeo entre el índice de Networkx y el de OMNeT++.
`self.__graph_path`	`Str`	`""`	Ruta del grafo elegido para la escena.

Atributo	Tipo	Default	Función
`self.__gname`	**Str**	**“”**	Nombre del grafo que coincidirá con el del archivo .ini en el proyecto de OMNeT++.
`self.__sname`	**Str**	**“”**	Nombre del grafo evitando ciertos caracteres y en minúsculas para el futuro tratamiento en la interfaz de OMNeT++, coincidirá con el nombre del directorio.
`self.__folder`	**Str**	**“”**	Aquí se guarda la ruta de trabajo por rendimiento.
`self.__xml`	**File**	**ruta**	Ruta para el archivo de configuración de capa 3 (IP) en formato xml.
`self.__Xmin = 0`	**Double**	**0**	Coordenada X de menor valor.
`self.__Xmax = 0`	**Double**	**0**	Coordenada X de mayor valor.
`self.__Ymin = 0`	**Double**	**0**	Coordenada Y de menor valor.
`self.__Ymax = 0`	**Double**	**0**	Coordenada Y de mayor valor.
`self.__path`	**OSPath**	**objeto**	Objeto de la clase OSPath que se encarga de gestionar los directorios de trabajo.
`self.set`	**OSSettings**	**objeto**	Objeto de la clase OSSetting, descrito más adelante, que se encarga de guardar y escribir los ajustes de cada escenario.
`self.color`	**[int]**	**[]**	Contiene un conjunto de enteros que representan el color de cada escenario.
`self.__rand`	**Bool**	**False**	Determina si el coloreado debe ser random o de lo contrario se ha especificado en la llamada uno concreto. Ofrece un mayor rendimiento a la hora de decidir el color en cada punto de acceso.

6.1.2. Métodos:

La mayor parte de los métodos son privados, puesto que se invocarán de forma automatizada por la propia aplicación. En el caso de que algún método contenga algún parámetro se adjuntará en su descripción una tabla con sus detalles, cuando uno de estos parámetros es obligatorio se representará como un asterisco “*” en el campo de valor por defecto (Default).

```
def __init__(self, graph_name, color = [], exprt = '' ):
```

En ***Python*** toda clase debe llevar el método **__init__**, se trata de un método privado de la propia clase, que funciona como constructor de la misma. Cuando se crea una instancia se invoca este método y es en él donde se definen e inician los atributos de la clase, en el caso de esta, recibe tres parámetros uno el nombre del grafo con la ruta incluida, el segundo es una lista de números que hacen referencia a los canales, los cuales deben estar en el mismo orden que los puntos de acceso y el último es un sufijo

que se añade al nombre del directorio de destino, con el fin de poder distinguir un escenario de otro en la interfaz que ofrece ***OMNeT++***.

Parámetro	Tipo	Default	Función
graph_name	**Str**	*	Contiene la ruta del grafo, con el nombre de este.
color	**[Int]**	**[]**	Lista de enteros con los canales de cada uno de los APs, se asignan en el orden de los APs.
exprt	**Str**	''	Sufijo para nombrar cada escenario y así diferenciarlos en distintos directorios dentro de OMNet++.

(*) Parámetro obligatorio en la llamada

```
def __addAp(self, index, node):
```

Este método se invoca una vez por cada nodo que actúa de punto de acceso, recibe el índice que referencia de ***NetworkX*** y el nodo que se añadirá en cada caso.

Parámetro	Tipo	Default	Función
index	**Int**	*	En este caso el índice que recibe es índice del grafo generado por Networkx que lo guardaremos como ID.
node	**OSAp**	*	Recibe un objeto de la clase OSAp, para añadirlo a la escena.

(*) Parámetro obligatorio en la llamada

```
def __addClient(self, index, node):
```

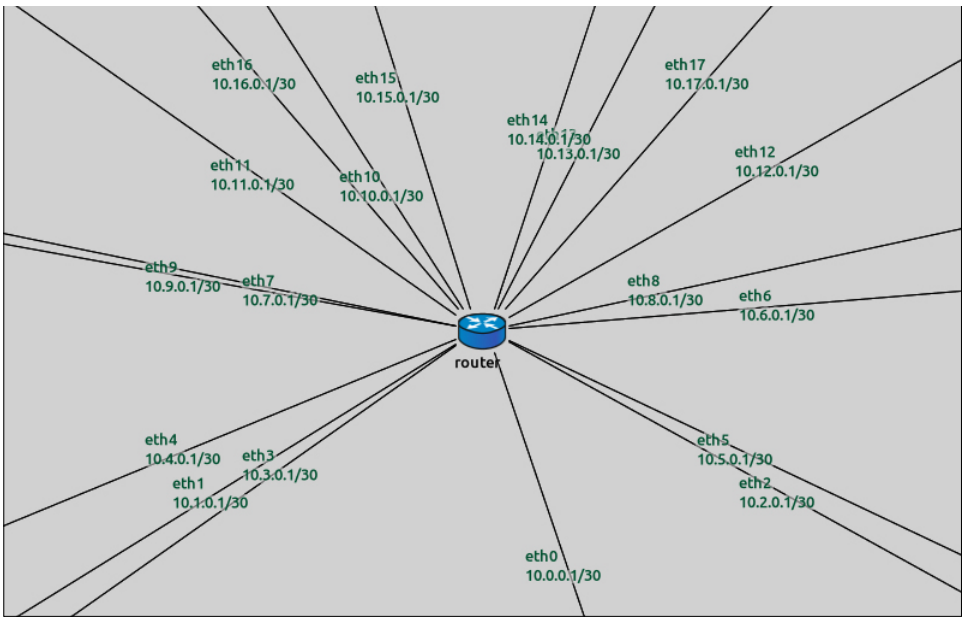
Este método se invoca una vez por cada nodo que actúa de cliente, recibe como parámetro el índice que referencia ***NetworkX*** y el nodo que se añadirá en cada caso.

Parámetro	Tipo	Default	Función
index	**Int**	*	En este caso el index que recibe es índice del grafo generado por networkx que lo guardaremos como ID.
node	**OSClient**	*	Recibe un objeto de la clase OSClient, para añadirlo a la escena.

(*) Parámetro obligatorio en la llamada

```
def __assoApClient(self):
```

Este método se encarga de generar toda la configuración de capa 3 (IP), el rango de direcciones elegido es el 10.0.0.0/8, donde el segundo byte representa la interfaz Ethernet del router al que está conectado un punto de acceso, por ejemplo en el caso de la interfaz 1 del router, el identificador de red sería 10.1.0.0/24, la dirección de esta interfaz sería la 10.1.0.1 y el resto de hosts que cuelgan de este punto de acceso, se les asignará una entre 10.1.0.2 y 10.1.0.254.



Captura 53: Configuración IP en interfaces Ethernet del router.

```
def __nodePrintToFile(self, name, index, node, outf):
```

Método privado, su función consiste en escribir los distintos nodos del escenario a generar en el archivo "*.ini*" de este mismo.

Parámetro	Tipo	Default	Función
name	**Str**	*	Nombre del módulo a escribir en el archivo .ini
index	**Int**	*	Es el índice del módulo para OMNeT++.
node	**OSNode** **OSAp** **OSClient**	*	Referencia al nodo que queremos escribir en el archivo .ini. Admite las tres clases OSNode, OSClient y OSAp.
outf	**File**	*	Fichero .ini de salida.

(*) Parámetro obligatorio en la llamada

```
def __genSimulation(self):
```

Este método se encarga de escribir toda la configuración en el archivo "*.ini*" así como copiar los archivos y directorios necesarios para que la simulación funcione.

```
def __fixPosition(self):
```

En el caso de que existan posiciones negativas tanto en X o Y, ***OMNeT++*** nos daría un error lanzando una advertencia, indicando que algún módulo está fuera del área de simulación, por esta razón fue necesario incluir este método, que se encarga de aplicar un offset en X e Y a partir de los atributos *self*.__Xmin y *self*.__Ymin.

```
def launchScene(self):
```

Este es el único método público de la clase, se encarga de leer el grafo e ir añadiendo los nodos en cada caso, para clientes y puntos de acceso, después de crear y configurar la simulación. El último paso sería ejecutar este método y se generarán los archivos necesarios para uno o más escenarios, en el directorio del proyecto destino.

6.2. Clase OSNode.

```
class OSNode(object):
```

Esta es la clase base para cualquier nodo. Contiene los atributos y parámetros que tienen en común los clientes con los puntos de acceso. De ella heredan otras dos clases que veremos más adelante.

6.2.1. Atributos:

Atributos de la clase ***OSNode***.

Atributo	Tipo	Default	Función
`self.id`	**Int**	*	Contiene los objetos que representan los clientes.
`self.mac`	**Str**	**0**	Número de clientes.
`self.pos`	**[Double]**	*	Contiene los objetos que representan los puntos de acceso.
`self.channel`	**Int**	**1**	Canal 802.11.
`self.ip`	**Str**	**'0.0.0.0'**	Dirección IP de cada nodo

(*) Parámetro obligatorio en la llamada

6.2.2. Métodos:

Esta clase aparte del constructor, sólo contiene un método público que se encarga de codificar las direcciones *MAC* de los nodos a partir de el número *ID* que se asigna con ***NetworkX***, con el fin de que sean únicas en el escenario.

```
def __init__(self, id, pos):
```

Constructor de la clase, recibe como parámetro la ID del nodo asignada por ***NetworkX*** y la posición en coordenadas (X,Y).

Parámetro	Tipo	Default	Función
id	**Int**	*	Recibe el ID asignado por Networkx.
pos	**[Double]**	*	Coordenadas (X,Y).

(*) Parámetro obligatorio en la llamada

```
def mapMAC(self, id, rng = “”):
```

Este método público genera la dirección *MAC* de cada nodo, la codificación la realiza en base a su índice asignado con ***NetworkX***.

Parámetro	Tipo	Default	Función
id	**Int**	*	Recibe el ID asignado por Networkx y lo utiliza para codificar la MAC en base a este número.
rng	**Str**	“”	Aquí se define el segundo byte de la dirección MAC, debe contener una cadena de caracteres con algún valor en hexadecimal comprendido entre el rango 00 → FF.

(*) Parámetro obligatorio en la llamada

6.3. Clase OSClient.

```
class OSClient(OSNode):
```

Esta clase es la encargada de gestionar la información de cada cliente, hereda todos los métodos y atributos de la clase ***OSNode***, añadiendo únicamente el atributo *self*.ap el cuál contiene la dirección *MAC* del punto de acceso con el que está asociado.

6.3.1. Atributos:

Esta clase añade un sólo atributo y hereda todos los de la clase ***OSNode***.

Atributo	Tipo	Default	Función
self.ap	**Int**	“”	Dirección MAC del punto de acceso al cual se conectará el cliente.

6.3.2. Métodos:

Esta clase contiene únicamente el método **__init__** que recibe dos parámetros la *ID* y coordenadas (X,Y), también hereda cualquier método de la clase ***OSNode***.

```
def __init__(self, id, pos):
```

Constructor de la clase, recibe como parámetro la *ID* del nodo asignada por ***NetworkX*** y la posición en coordenadas (X,Y).

Parámetro	Tipo	Default	Función
id	**Int**	*	Recibe el ID asignado por NetworkX.
pos	**[Double]**	*	Coordenadas (X,Y).

(*) Parámetro obligatorio en la llamada

6.4. Clase OSAp.

```
class OSAp(OSNode):
```

Esta clase contiene la información de cada punto de acceso, hereda de la clase ***OSNode*** todos sus atributos y métodos, añadiendo algunos característicos de este tipo de nodo.

6.4.1. Atributos:

Atributo	Tipo	Default	Función
self.listCli	**[Int]**	*	Lista que contiene los números ID de los clientes que tienen asociados.
self.channel	**Int**	*	Número de canal de la interfaz Wifi.
self.ethmac	**Str**	**MAC**	Dirección MAC de la interfaz Ethernet.

(*) Parámetro obligatorio en la llamada

6.4.2. Métodos:

Esta clase sólo contiene método **__init__** que recibe dos parámetros la *ID* y coordenadas (X,Y), también hereda cualquier método de la clase ***OSNode***.

```
def __init__(self, id, pos):
```

Constructor de la clase, recibe como parámetro la *ID* del nodo asignada por ***Networkx*** y la posición en coordenadas (X,Y).

Parámetro	Tipo	Default	Función
id	**Int**	*	Recibe el ID asignado por Networkx.

Parámetro	Tipo	Default	Función
pos	[Double]	*	Coordenadas (X,Y).

(*) Parámetro obligatorio en la llamada

6.5. Clase OSSettings.

```
class OSSettings(object):
```

Esta clase es la encargada de gestionar todos los pequeños ajustes del escenario en cuestión. Una vez esté creada la instancia del objeto ***OSScene***, podremos acceder a estos ajustes a través del atributo ***OSScene.set*** y configurar todos los parámetros de las interfaces, así como el número de repeticiones o el tiempo máximo que deseamos ejecutar la simulación.

6.5.1. Atributos:

La mayoría de atributos de esta clase son diccionarios de ***Python***, por lo que vendrán desglosados por conjuntos. Estos mapas guardan los ajustes del escenario agrupándolos en modelos, por ejemplo todos los ajustes de Ethernet están en el mismo diccionario.

Atributo	Tipo	Default	Función
self.timeLimit	Str	""	Tiempo máximo de ejecución, puede estar expresado en milisegundos (ms), segundos (s), minutos (m), horas (h).
self.repeat	Int	1	Número de repeticiones de la simulación.
self.cServer	Bool	False	Define un servidor TCP/UDP central, los hosts únicamente funcionan como clientes.
self.udp	{%}	--	Diccionario UDP
↳['videoSize']	Str	"1000MiB"	Tamaño de vídeo a enviar.
↳['sendInterval']	Str	"10ms"	Intervalo de envío, puede ser expresado en milisegundos (ms), segundos (s).
↳['packetLen']	Str	"1000B"	Longitud por paquete, puede ser expresado en bits (b) o bytes (B).
↳['lambda']	Str	""	Tasa de llegadas al servicio, si no se especifica una tasa todos los clientes comenzarán el envío en t = 0, puede ser expresado en milisegundos (ms), segundos (s), minutos (m), horas (h).
↳['servers']	[Int]	[]	Lista que contiene los números ID de los nodos que actuarán como servidores, si no se especifica el host[0] hará de servidor.

Atributo	Tipo	Default	Función
↳['clients']	[Int]	[]	Lista que contiene los números ID de los nodos que actuarán como clientes, si no se especifica todos los restantes actuarán de clientes.
self.tcp	{%}	--	**Diccionario TCP**
↳['active']	Bool	True	Determina el modo de sesión TCP si es activo el cliente inicia la sesión y en caso contrario es el servidor el que la inicia.
↳['sendBytes']	Str	"1000MiB"	Determina el tamaño del paquete que quiere enviar cada cliente.
↳['servers']	[Int]	[]	Lista que contiene los números ID de los nodos que actuarán como servidores, si no se especifica el host[0] hará de servidor.
↳['clients']	[Int]	[]	Lista que contiene los números ID de los nodos que actuarán como clientes, si no se especifica todos los restantes actuarán de clientes.
self.ping	{%}	--	**Diccionario PING**
↳['background']	Bool	False	Indica si el tráfico ICMP está presente en las simulaciones TCP y UDP.
↳['packetSize']	Str	""	Longitud del paquete ICMP.
↳['printPing']	Bool	False	Indica si se imprimen en la consola de OMNeT++ los paquetes de respuesta, como si de una bash se tratase.
self.eth	{%}	--	**Diccionario Ethernet**
↳['duplexMode']	Bool	True	Indica el tipo de transmisión de datos de Ethernet duplex o simplex.
↳['mtu']	Str	"1500B"	MTU, unidad de transmisión máxima de Ethernet.
↳['promiscuous']	Bool	False	Determina si la interfaz Ethernet recibe paquetes de otras interfaces.
self.wlan	{%}	--	**Diccionario Wifi**
↳['opMode']	Str	""	Modo de operación 802.11x, puede ser "b", "p", "g" o "n". Si se deja en blanco, tomará el valor por defecto de OMNeT++, "g".
↳['bandwidth']	Str	""	Ancho de banda de cada canal. Si se deja en blanco, tomará el valor por defecto de OMNeT++, 2MHz.
↳['carrierFrequency']	Str	""	Frecuencia de trabajo de la portadora, Si se deja en blanco, tomará el valor por defecto de OMNeT++ que es la más común, 2,4 GHz.
↳['txPower']	Str	""	Determina la potencia de transmisión de la antena expresada en W, el valor por defecto de OMNeT++ es 20mW.

Atributo	Tipo	Default	Función
↳['sensitivity']	**Str**	**""**	Determina la sensibilidad de recepción en antenas expresada en W, el valor por defecto de OMNeT++ es de -85dBm
↳['snirThreshold']	**Str**	**""**	Umbral funcionamiento de S/N, relación señal ruido, por defecto en OMNeT++ 4dB.
↳['gain']	**Str**	**""**	Ganancia de la antenas, tanto transmisor como receptor, por defecto en OMNeT++ 0dB.

(*) Parámetro obligatorio en la llamada.
(%) Contiene varios tipos de datos.

6.5.2. Métodos:

Los métodos de esta clase deben ser accesibles desde la clase ***OSScene*** que la contiene por lo que deben ser todos públicos, estos se encargan de escribir la configuración en base a los parámetros elegidos previamente por el usuario.

```
def __init__(self):
```

Constructor de la clase, no recibe ningún parámetro.

```
def ethSettings(self, oini):
```

Este método se encarga de escribir, en base a los atributos "*self.eth*" expuestos anteriormente, todos los ajustes de Ethernet, el modo duplex, la MTU y el modo promiscuo. Estos cambios afectarán a todos los nodos que contengan esta interfaz.

Parámetro	Tipo	Default	Función
oini	**File**	*	Recibe como parámetro el archivo .ini de salida.

(*) Parámetro obligatorio en la llamada

```
def wlanSettings(self, oini):
```

Este método se encarga de escribir, en base a los atributos "*self*.wlan" expuestos anteriormente, todos los ajustes de ***Wifi***. Estos cambios afectarán a todos los nodos que contengan esta interfaz.

Parámetro	Tipo	Default	Función
oini	**File**	*	Recibe como parámetro el archivo .ini de salida.

(*) Parámetro obligatorio en la llamada

```
def configSettings(self, oini):
```

Este método se encarga de añadir al archivo "*.ini*" las configuraciones aisladas de *TCP*, *UDP* e *ICMP*, es decir es el encargado de configurar la capa de aplicación del escenario, pudiendo elegir entre una opción u otra a la hora de simular en ***OMNeT++***.

Parámetro	Tipo	Default	Función
oini	File	*	Recibe como parámetro el archivo .ini de salida.

(*) Parámetro obligatorio en la llamada

6.6. Clase OSPath.

```
class OSPath(object):
```

Esta aplicación depende de dos directorios de trabajo. Uno es donde tenemos alojado el proyecto principal de ***OMNeT++*** y el otro es el propio directorio de ***OGSimulation***. Estas dos ubicaciones contienen subdirectorios que se solicitarán a través de varios métodos que expondremos más adelante.

Fue necesario incluir este objeto en la escena con el fin de facilitar la forma de uso de cara al usuario final, por ello cuando se crea una instancia de ***OSScene*** se solicitan estas dos ubicaciones a través de la interfaz gráfica que proporciona el protocolo *X11* y se guardan en un archivo de texto llamado "*path*". En el caso de existir este fichero las ubicaciones no se solicitarán y se leerán directamente de este.

6.6.1. Atributos:

Para acceder a las ubicaciones principales disponemos de dos atributos públicos, uno para cada caso. Los subdirectorios de estos serán accesibles a través de métodos.

Atributo	Tipo	Default	Función
self.opp	Str	*	Contiene el path del directorio principal de OMNeT++, es requerido en la llamada y se solicita por interfaz gráfica a través del protocolo X11.
self.OGS	Str	*	Contiene el path del directorio principal de la propia aplicación OGSimulation, es requerido en la llamada y se solicita por interfaz gráfica a través del protocolo X11.

(*) Parámetro obligatorio en la llamada

6.6.2. Métodos:

Para acceder al directorio raíz de ambas ubicaciones disponemos de dos atributos, pero para acceder a los demás subdirectorios se solicitan mediante los siguientes métodos.

```
def getOPPSimulationPath(self):
```

Devuelve una cadena de caracteres que contiene la ubicación del subdirectorio “Simulations” contenido en el directorio principal del proyecto de ***OMNeT++***.

```
def getOPPImagesPath(self):
```

Devuelve una cadena de caracteres que contiene la ubicación del subdirectorio “Images” contenido en el directorio principal del proyecto de ***OMNeT++***.

```
def getOGSTemplatesPath(self):
```

Devuelve una cadena de caracteres que contiene la ubicación del subdirectorio “Templates” contenido en el directorio principal del proyecto de ***OGSimulation***.

```
def getOGSImagesPath(self):
```

Devuelve una cadena de caracteres que contiene la ubicación del subdirectorio “Images” contenido en el directorio principal del proyecto de ***OGSimulation***.

7. REFERENCIAS EXTERNAS.

7.1. Bibliografía

[1] E. Au, M. Cheong, C. Ngo, C. Cordeiro, W. Zhuang, "The future of Wi-Fi - Guest Editorial", IEEE Communications Magazine , vol.52, no.11, pp.20,21, 2014.

[2] P. Dalal, M. Sarkar, K. Dasgupta, N. Kothar, "Link Layer Correction Techniques and Impact on TCP's Performance in IEEE 802.11 Wireless Network", Communications and Network, vol.6, no.2, ID 45714, 2014.

[3] R. Liao, B. Bellalta, M. Oliver, Z. Niu, "MU-MIMO MAC Protocols for Wireless Local Area Networks: A Survey," IEEE Communications Surveys & Tutorials (pendiente de publicación

[4] Marsa-Maestre, I.; Lopez-Carmona, M.A.; Velasco, J.R.; de la Hoz, E. Avoiding the prisoner's dilemma in auction-based negotiations for highly rugged utility spaces. In Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Toronto, ON, Canada, 10–14 May 2010; pp. 425–432.

[5] Marsa-Maestre, I.; Lopez-Carmona, M.A.; Velasco, J.R.; Ito, T.; Klein, M.; Fujita, K. Balancing utility and deal probability for auction-based negotiations in highly nonlinear utility spaces. In Proceedings of the 21st International Jont Conference on Artifical Intelligence (IJCAI), Pasadena, CA, USA, 11–17 July 2009; pp. 214–219.

[7] M. Achanta, "Method and apparatus for least congested channel scan for wireless access points", US Patent App. 10/959,446, 2006.

7.2. Para consultar on-line.

[1] Automated negotiation for resource assignment in wireless surveillance sensor networks. Enrique de la Hoz, José Manuel Giménez-Guzmán, Iván Marsá-Maestre, David Orden. Sensors vol. 15 (11), pp. 29547-29568.

[2] Documentación de OMNeT++ simulador de eventos discretos orientado a redes de comunicación https://omnetpp.org/doc/omnetpp/manual/

[3] Documentación de INET framework para OMNeT++ que contiene todos los modelos de los estándares de IEEE https://inet.omnetpp.org

[4] Documentación de NetworkX framework escrito en Python para estudio de redes de comunicaciones con técnicas basadas en teoría de grafos https://networkx.github.io

[5] Sitio web oficial de estándares ofrecidos por el equipo de investigación IEEE (Institute of Electrical and Electronics Engineers) https://standards.ieee.org

ANEXO I: Coloreado del escenario simulado.

AP	WORST	LCCS	SA
0	6	9	6
1	6	1	11
2	6	8	1
3	6	11	6
4	6	8	3
5	6	2	9
6	6	11	6
7	6	4	9
8	6	5	3
9	6	1	1
10	6	1	7
11	6	7	4
12	6	8	11
13	6	3	5
14	6	11	1
15	6	9	11
16	6	5	1
17	6	5	8

ANEXO II: Repeticiones del escenario simulado.

REPETICIÓN 0

WORST	7640000	Bytes	LCCS	9964000	Bytes	SA	11541000	Bytes
Host	**Value**	**loss**	**Host**	**Value**	**loss**	**Host**	**Value**	**loss**
109	0	NaN	109	0	NaN	109	0	NaN
110	952000	8,46%	110	1423000	4,56%	110	1453000	2,61%
111	967000	7,38%	111	1430000	4,22%	111	1450000	2,95%
112	0	NaN	112	0	NaN	112	0	NaN
113	960000	6,25%	113	1422000	3,98%	113	1430000	3,18%
114	962000	7,14%	114	1423000	4,30%	114	1440000	3,23%
115	970000	7,27%	115	1412000	5,74%	115	1443000	3,74%
116	949000	7,95%	116	1427000	4,10%	116	1439000	3,10%
117	944000	8,08%	117	0	NaN	117	1441000	2,70%
118	936000	8,24%	118	1427000	3,78%	118	1445000	2,56%

WORST	8268000	Bytes	LCCS	10421000	Bytes	SA	11899000	Bytes
Host	**Value**		**Host**	**Value**		**Host**	**Value**	
49	0	Bytes	49	0	Bytes	49	0	Bytes
50	1040000	Bytes	50	1491000	Bytes	50	1492000	Bytes
51	1044000	Bytes	51	1493000	Bytes	51	1494000	Bytes
52	0	Bytes	52	0	Bytes	52	0	Bytes
53	1024000	Bytes	53	1481000	Bytes	53	1477000	Bytes
54	1036000	Bytes	54	1487000	Bytes	54	1488000	Bytes
55	1046000	Bytes	55	1498000	Bytes	55	1499000	Bytes
56	1031000	Bytes	56	1488000	Bytes	56	1485000	Bytes
57	1027000	Bytes	57	0	Bytes	57	1481000	Bytes
58	1020000	Bytes	58	1483000	Bytes	58	1483000	Bytes

REPETICIÓN 1

WORST	6677000	Bytes	LCCS	8680000	Bytes	SA	8740000	Bytes
Host	**Value**	**loss**	**Host**	**Value**	**loss**	**Host**	**Value**	**loss**
109	1116000	7,00%	109	0	NaN	109	0	NaN
110	1107000	9,19%	110	1448000	3,27%	110	1448000	3,34%
111	0	NaN	111	0	NaN	111	0	NaN
112	1112000	8,33%	112	1445000	3,22%	112	1455000	2,48%
113	0	NaN	113	0	NaN	113	0	NaN
114	0	NaN	114	0	NaN	114	0	NaN
115	1126000	7,70%	115	1444000	3,54%	115	1461000	2,47%
116	1103000	8,31%	116	1442000	3,16%	116	1457000	2,21%
117	0	NaN	117	1453000	2,15%	117	1454000	2,09%
118	1113000	7,94%	118	1448000	2,88%	118	1465000	1,94%

WORST	7264000	Bytes	LCCS	8952000	Bytes	SA	8957000	Bytes
Host	**Value**		**Host**	**Value**		**Host**	**Value**	

49	1200000	Bytes	49	0	Bytes	49	0	Bytes
50	1219000	Bytes	50	1497000	Bytes	50	1498000	Bytes
51	0	Bytes	51	0	Bytes	51	0	Bytes
52	1213000	Bytes	52	1493000	Bytes	52	1492000	Bytes
53	0	Bytes	53	0	Bytes	53	0	Bytes
54	0	Bytes	54	0	Bytes	54	0	Bytes
55	1220000	Bytes	55	1497000	Bytes	55	1498000	Bytes
56	1203000	Bytes	56	1489000	Bytes	56	1490000	Bytes
57	0	Bytes	57	1485000	Bytes	57	1485000	Bytes
58	1209000	Bytes	58	1491000	Bytes	58	1494000	Bytes

REPETICIÓN 2

WORST	8371000	Bytes	**LCCS**	8602000	Bytes	**SA**	10049000	Bytes
Host	**Value**	**loss**	**Host**	**Value**	**loss**	**Host**	**Value**	**loss**
109	0	NaN	109	0	NaN	109	1455000	2,15%
110	1424000	4,81%	110	0	NaN	110	0	NaN
111	1423000	5,01%	111	0	NaN	111	1455000	2,81%
112	1354000	4,45%	112	1416000	2,21%	112	1460000	1,62%
113	0	NaN	113	0	NaN	113	0	NaN
114	1419000	4,96%	114	1443000	3,35%	114	1454000	2,35%
115	1359000	4,56%	115	1407000	2,22%	115	1296000	1,59%
116	0	NaN	116	1445000	2,76%	116	0	NaN
117	1392000	4,66%	117	1445000	2,96%	117	1461000	1,81%
118	0	NaN	118	1446000	3,47%	118	1468000	1,61%

WORST	8788000	Bytes	**LCCS**	8853000	Bytes	**SA**	10254000	Bytes
Host	**Value**		**Host**	**Value**		**Host**	**Value**	
49	0	Bytes	49	0	Bytes	49	1487000	Bytes
50	1496000	Bytes	50	0	Bytes	50	0	Bytes
51	1498000	Bytes	51	0	Bytes	51	1497000	Bytes
52	1417000	Bytes	52	1448000	Bytes	52	1484000	Bytes
53	0	Bytes	53	0	Bytes	53	0	Bytes
54	1493000	Bytes	54	1493000	Bytes	54	1489000	Bytes
55	1424000	Bytes	55	1439000	Bytes	55	1317000	Bytes
56	0	Bytes	56	1486000	Bytes	56	0	Bytes
57	1460000	Bytes	57	1489000	Bytes	57	1488000	Bytes
58	0	Bytes	58	1498000	Bytes	58	1492000	Bytes

REPETICIÓN 3

WORST	6106000	Bytes	**LCCS**	11357000	Bytes	**SA**	11550000	Bytes
Host	**Value**	**loss**	**Host**	**Value**	**loss**	**Host**	**Value**	**loss**
109	0	NaN	109	1428000	3,45%	109	1443000	2,37%
110	1228000	5,32%	110	1430000	4,41%	110	1457000	2,41%
111	1213000	3,65%	111	1393000	5,50%	111	1437000	3,17%
112	0	NaN	112	1419000	3,86%	112	1452000	1,96%
113	0	NaN	113	0	NaN	113	0	NaN

114	0	NaN	114	1413000	4,66%	114	1432000	3,70%
115	0	NaN	115	0	NaN	115	0	NaN
116	1215000	5,59%	116	1417000	5,09%	116	1429000	4,16%
117	1227000	5,76%	117	1429000	4,61%	117	1449000	3,21%
118	1223000	3,47%	118	1428000	3,90%	118	1451000	2,62%
WORST	6412000	Bytes	**LCCS**	11884000	Bytes	**SA**	11901000	Bytes
Host	**Value**		**Host**	**Value**		**Host**	**Value**	
49	0	Bytes	49	1479000	Bytes	49	1478000	Bytes
50	1297000	Bytes	50	1496000	Bytes	50	1493000	Bytes
51	1259000	Bytes	51	1474000	Bytes	51	1484000	Bytes
52	0	Bytes	52	1476000	Bytes	52	1481000	Bytes
53	0	Bytes	53	0	Bytes	53	0	Bytes
54	0	Bytes	54	1482000	Bytes	54	1487000	Bytes
55	0	Bytes	55	0	Bytes	55	0	Bytes
56	1287000	Bytes	56	1493000	Bytes	56	1491000	Bytes
57	1302000	Bytes	57	1498000	Bytes	57	1497000	Bytes
58	1267000	Bytes	58	1486000	Bytes	58	1490000	Bytes

REPETICIÓN 4

WORST	6986000	Bytes	**LCCS**	8687000	Bytes	**SA**	10172000	Bytes
Host	**Value**	**loss**	**Host**	**Value**	**loss**	**Host**	**Value**	**loss**
109	0	NaN	109	0	NaN	109	0	NaN
110	1386000	5,00%	110	0	NaN	110	0	NaN
111	1414000	5,48%	111	1454000	2,87%	111	1465000	2,14%
112	0	NaN	112	1449000	2,03%	112	1447000	3,08%
113	1391000	5,31%	113	1441000	2,70%	113	1460000	1,95%
114	0	NaN	114	0	NaN	114	1455000	2,09%
115	1390000	6,84%	115	1456000	2,54%	115	0	NaN
116	0	NaN	116	0	NaN	116	1450000	2,49%
117	0	NaN	117	1447000	2,03%	117	1450000	2,29%
118	1405000	4,68%	118	1440000	2,24%	118	1445000	3,02%
WORST	7390000	Bytes	**LCCS**	8901000	Bytes	**SA**	10426000	Bytes
Host	**Value**		**Host**	**Value**		**Host**	**Value**	
49	0	Bytes	49	0	Bytes	49	0	Bytes
50	1459000	Bytes	50	0	Bytes	50	0	Bytes
51	1496000	Bytes	51	1497000	Bytes	51	1497000	Bytes
52	0	Bytes	52	1479000	Bytes	52	1493000	Bytes
53	1469000	Bytes	53	1481000	Bytes	53	1489000	Bytes
54	0	Bytes	54	0	Bytes	54	1486000	Bytes
55	1492000	Bytes	55	1494000	Bytes	55	0	Bytes
56	0	Bytes	56	0	Bytes	56	1487000	Bytes
57	0	Bytes	57	1477000	Bytes	57	1484000	Bytes
58	1474000	Bytes	58	1473000	Bytes	58	1490000	Bytes

REPETICIÓN 5

WORST	8146000	Bytes	LCCS	9810000	Bytes	SA	8708000	Bytes
Host	**Value**	**loss**	**Host**	**Value**	**loss**	**Host**	**Value**	**loss**
109	0	NaN	109	0	NaN	109	0	NaN
110	1361000	8,66%	110	1413000	5,17%	110	1453000	2,74%
111	0	NaN	111	1418000	3,99%	111	1435000	3,63%
112	0	NaN	112	1315000	3,80%	112	0	NaN
113	1358000	9,16%	113	1420000	5,02%	113	1455000	2,68%
114	1359000	7,11%	114	0	NaN	114	1451000	2,42%
115	1352000	8,96%	115	1414000	4,78%	115	0	NaN
116	1366000	8,81%	116	1415000	5,48%	116	1467000	2,07%
117	1350000	7,98%	117	0	NaN	117	0	NaN
118	0	NaN	118	1415000	4,84%	118	1447000	2,95%

WORST	8898000	Bytes	LCCS	10298000	Bytes	SA	8954000	Bytes
Host	**Value**		**Host**	**Value**		**Host**	**Value**	
49	0	Bytes	49	0	Bytes	49	0	Bytes
50	1490000	Bytes	50	1490000	Bytes	50	1494000	Bytes
51	0	Bytes	51	1477000	Bytes	51	1489000	Bytes
52	0	Bytes	52	1367000	Bytes	52	0	Bytes
53	1495000	Bytes	53	1495000	Bytes	53	1495000	Bytes
54	1463000	Bytes	54	0	Bytes	54	1487000	Bytes
55	1485000	Bytes	55	1485000	Bytes	55	0	Bytes
56	1498000	Bytes	56	1497000	Bytes	56	1498000	Bytes
57	1467000	Bytes	57	0	Bytes	57	0	Bytes
58	0	Bytes	58	1487000	Bytes	58	1491000	Bytes

REPETICIÓN 6

WORST	8266000	Bytes	LCCS	9746000	Bytes	SA	8718000	Bytes
Host	**Value**	**loss**	**Host**	**Value**	**loss**	**Host**	**Value**	**loss**
109	1397000	6,37%	109	1429000	4,16%	109	0	NaN
110	0	NaN	110	1407000	3,70%	110	1466000	1,35%
111	0	NaN	111	0	NaN	111	0	NaN
112	1378000	5,36%	112	1397000	4,71%	112	1453000	1,89%
113	0	NaN	113	1263000	3,59%	113	0	NaN
114	1370000	8,24%	114	1420000	4,95%	114	1455000	2,94%
115	1389000	5,45%	115	0	NaN	115	1458000	2,08%
116	1381000	7,69%	116	1430000	4,41%	116	1470000	1,80%
117	0	NaN	117	0	NaN	117	0	NaN
118	1351000	6,05%	118	1400000	3,51%	118	1416000	1,53%

WORST	8844000	Bytes	LCCS	10169000	Bytes	SA	8890000	Bytes
Host	**Value**		**Host**	**Value**		**Host**	**Value**	
49	1492000	Bytes	49	1491000	Bytes	49	0	Bytes
50	0	Bytes	50	1461000	Bytes	50	1486000	Bytes

51	0	Bytes	51	0	Bytes	51	0	Bytes
52	1456000	Bytes	52	1466000	Bytes	52	1481000	Bytes
53	0	Bytes	53	1310000	Bytes	53	0	Bytes
54	1493000	Bytes	54	1494000	Bytes	54	1499000	Bytes
55	1469000	Bytes	55	0	Bytes	55	1489000	Bytes
56	1496000	Bytes	56	1496000	Bytes	56	1497000	Bytes
57	0	Bytes	57	0	Bytes	57	0	Bytes
58	1438000	Bytes	58	1451000	Bytes	58	1438000	Bytes

REPETICIÓN 7

WORST	7198000	Bytes	**LCCS**	9975000	Bytes	**SA**	11503000	Bytes
Host	**Value**	**loss**	**Host**	**Value**	**loss**	**Host**	**Value**	**loss**
109	911000	10,60%	109	1430000	3,25%	109	1452000	2,09%
110	921000	10,50%	110	1425000	3,78%	110	1448000	2,56%
111	0	NaN	111	0	NaN	111	0	NaN
112	920000	10,16%	112	1429000	3,90%	112	1443000	3,22%
113	0	NaN	113	0	NaN	113	1447000	1,96%
114	925000	8,23%	114	0	NaN	114	1426000	3,58%
115	752000	10,37%	115	1423000	4,05%	115	1419000	4,96%
116	935000	9,31%	116	1433000	3,83%	116	1439000	3,29%
117	913000	11,96%	117	1437000	4,01%	117	1429000	4,61%
118	921000	9,08%	118	1398000	5,16%	118	0	NaN

WORST	8000000	Bytes	**LCCS**	10390000	Bytes	**SA**	11894000	Bytes
Host	**Value**		**Host**	**Value**		**Host**	**Value**	
49	1019000	Bytes	49	1478000	Bytes	49	1483000	Bytes
50	1029000	Bytes	50	1481000	Bytes	50	1486000	Bytes
51	0	Bytes	51	0	Bytes	51	0	Bytes
52	1024000	Bytes	52	1487000	Bytes	52	1491000	Bytes
53	0	Bytes	53	0	Bytes	53	1476000	Bytes
54	1008000	Bytes	54	0	Bytes	54	1479000	Bytes
55	839000	Bytes	55	1483000	Bytes	55	1493000	Bytes
56	1031000	Bytes	56	1490000	Bytes	56	1488000	Bytes
57	1037000	Bytes	57	1497000	Bytes	57	1498000	Bytes
58	1013000	Bytes	58	1474000	Bytes	58	0	Bytes

REPETICIÓN 8

WORST	5463000	Bytes	**LCCS**	5641000	Bytes	**SA**	5729000	Bytes
Host	**Value**	**loss**	**Host**	**Value**	**loss**	**Host**	**Value**	**loss**
109	1369000	7,25%	109	1417000	4,39%	109	1429000	3,58%
110	0	NaN	110	1418000	4,58%	110	1442000	3,35%
111	0	NaN	111	0	NaN	111	0	NaN
112	1357000	8,99%	112	1415000	4,07%	112	1428000	3,84%
113	1383000	7,68%	113	1425000	4,49%	113	1432000	3,96%
114	1368000	7,51%	114	1414000	4,72%	114	1427000	4,16%
115	1367000	8,56%	115	1417000	5,34%	115	1435000	4,21%

116	0	NaN	116	0	NaN	116	0	NaN
117	1356000	8,99%	117	1414000	4,46%	117	1434000	3,04%
118	1372000	7,67%	118	1396000	6,31%	118	1433000	3,63%
WORST	10415000	Bytes	**LCCS**	11886000	Bytes	**SA**	11903000	Bytes
Host	**Value**		**Host**	**Value**		**Host**	**Value**	
49	1476000	Bytes	49	1482000	Bytes	49	1482000	Bytes
50	0	Bytes	50	1486000	Bytes	50	1492000	Bytes
51	0	Bytes	51	0	Bytes	51	0	Bytes
52	1491000	Bytes	52	1475000	Bytes	52	1485000	Bytes
53	1498000	Bytes	53	1492000	Bytes	53	1491000	Bytes
54	1479000	Bytes	54	1484000	Bytes	54	1489000	Bytes
55	1495000	Bytes	55	1497000	Bytes	55	1498000	Bytes
56	0	Bytes	56	0	Bytes	56	0	Bytes
57	1490000	Bytes	57	1480000	Bytes	57	1479000	Bytes
58	1486000	Bytes	58	1490000	Bytes	58	1487000	Bytes

REPETICIÓN 9

WORST	4178000	Bytes	**LCCS**	4361000	Bytes	**SA**	5784000	Bytes
Host	**Value**	**loss**	**Host**	**Value**	**loss**	**Host**	**Value**	**loss**
109	0	NaN	109	0	NaN	109	1401000	2,78%
110	1364000	6,25%	110	1455000	0,82%	110	1431000	3,31%
111	1381000	7,56%	111	1471000	1,61%	111	1430000	4,03%
112	1372000	5,90%	112	0	NaN	112	1450000	2,16%
113	0	NaN	113	0	NaN	113	0	NaN
114	0	NaN	114	0	NaN	114	0	NaN
115	1381000	4,82%	115	0	NaN	115	1462000	2,01%
116	1391000	6,64%	116	1458000	2,28%	116	1440000	3,23%
117	0	NaN	117	1445000	1,97%	117	1436000	3,30%
118	1406000	6,08%	118	1458000	2,54%	118	1446000	3,47%
WORST	8845000	Bytes	**LCCS**	7424000	Bytes	**SA**	11856000	Bytes
Host	**Value**		**Host**	**Value**		**Host**	**Value**	
49	0	Bytes	49	0	Bytes	49	1441000	Bytes
50	1455000	Bytes	50	1467000	Bytes	50	1480000	Bytes
51	1494000	Bytes	51	1495000	Bytes	51	1490000	Bytes
52	1458000	Bytes	52	0	Bytes	52	1482000	Bytes
53	0	Bytes	53	0	Bytes	53	0	Bytes
54	0	Bytes	54	0	Bytes	54	0	Bytes
55	1451000	Bytes	55	0	Bytes	55	1492000	Bytes
56	1490000	Bytes	56	1492000	Bytes	56	1488000	Bytes
57	0	Bytes	57	1474000	Bytes	57	1485000	Bytes
58	1497000	Bytes	58	1496000	Bytes	58	1498000	Bytes